27-я задача | КЕГЭ

Обработка числовых последовательностей

Автор:
Леонид Шастин

Оглавление

Что представляет из себя 27-я задача?

Задача №27, закрепленная за последним номером, традиционно считается самой сложной. Суть состоит в обработке числовых последовательностей, работе с различными свойствами чисел, комбинаторикой, логикой. Требуемые навыки: мат. база, ЯП - хорошее владение массивами, списками, словарями, другими функциями, умение логически рассуждать и писать программу с нуля под определенные условия. Как показывает практика, если прорешать сотню различных задач, всё кажется довольно простым. Это вполне реально сделать при подготовке к КЕГЭ, тем более с учетом того, что многие задачи идейно и структурно похожи друг на друга

Какие виды 27-х задач существуют?

1) поиск количества пар/групп элементов по определенным критериям
2) поиск суммы/произведения элементов по определенным критериям
3) обработка подпоследовательностей по определенным критериям
4) распределение элементов по группам по определенным критериям
5) задачи с учетом | i - j | >= y расстояния и других мелочей
6) исключительно экзотические задачи (прогрессии и т. п.)

Поговорим же вкратце обо всех этих типах задач...

Поиск количества пар/групп (некие условия)

27 (№ 2672) (Д.В. Богданов) Имеется набор данных, состоящий из положительных целых чисел. Необходимо определить количество пар элементов (a_i, a_j) этого набора, в которых $1 \le i < j \le N$ и произведение элементов кратно 6.

Входные данные. Даны два входных файла (файл A и файл B), каждый из которых содержит в первой строке количество чисел N ($1 \le N \le 100000$). Каждая из следующих N строк содержит одно натуральное число, не превышающее 10 000.

Пример входного файла:

```
4
7
5
6
12
```

Для указанных входных данных количество подходящих пар должно быть равно 5. В приведённом наборе из 4 чисел имеются пять пар (7, 6), (5, 6), (7, 12), (5, 12), (6, 12), произведение элементов которых кратно 6.

В ответе укажите два числа: сначала количество подходящих пар для файла A, затем для файла B.

Работаем с “прямыми” и “обратными” остатками, речь пойдёт о кратности элементов 6. Постепенное знакомство с массивами и усложнение

Поиск суммы/произведения (некие условия)

27 (№ 2679) (А. Жуков) Имеется набор данных, состоящий из целых чисел. Необходимо определить максимальное произведение подпоследовательности, состоящей из одного или более идущих подряд элементов.

Входные данные. Даны два входных файла (файл A и файл B), каждый из которых содержит в первой строке количество чисел N ($1 \le N \le 100000$). Каждая из следующих N строк содержит одно натуральное число, не превышающее по модулю 100.

Пример входного файла:

```
7
2
3
-2
-3
-1
4
6
```

Для указанных входных данных наибольшее произведение равно 72. Его можно получить для последовательности -3 -1 4 6.

В ответе укажите два числа: сначала искомое значение для файла A, затем для файла B.

Работаем с поиском максимальных и минимальных элементов, совместной их обработкой, увеличивающимися постепенно суммами и произведениями

Поиск длины подпоследовательности (некие условия)

27 (№ 4281) (демо-2022) Дана последовательность из N натуральных чисел. Рассматриваются все её непре-рывные подпоследовательности, такие что сумма элементов каждой из них кратна k = 43. Найдите среди них подпоследовательность с максимальной суммой, определите её длину. Если таких подпоследовательностей найдено несколько, в ответе укажите количество элементов самой короткой из них.

Входные данные. Даны два входных файла (файл A и файл B), каждый из которых содержит в первой строке количество чисел N ($2 \le N \le 10^8$). Каждая из следующих N строк содержит натуральное число, не превышающее 10000.

Пример входного файла:

```
7
21
13
9
19
17
26
95
```

В этом наборе можно выбрать последовательности 21+13+9 (сумма 43) и 17+26 (сумма 43). Самая короткая из них, 17 + 26, имеет длину 2. Ответ: 2.

В ответе укажите два числа: сначала искомое значение для файла A, затем для файла B.

Демоверсия | Работаем с массивами, списками, множествами, словарями

Распределением по группам (некие условия)

27 (№ 3149) Дана последовательность, которая состоит из троек натуральных чисел. Необходимо распределить все числа на три группы, при этом в каждую группу должно попасть ровно одно число из каждой исходной тройки. Сумма всех чисел как в первой, так и во второй группе должна быть чётной. Определите минимально возможную сумму всех чисел в третьей группе.

Входные данные. Даны два входных файла (файл A и файл B), каждый из которых содержит в первой строке количество чисел N ($1 \le N \le 100000$). Каждая из следующих N строк содержит три натуральных числа, не превышающих 10000.

Пример входного файла:

```
3
1 2 3
8 11 4
6 9 7
```

Для указанных данных искомая сумма равна 11, она соответствует такому распределению чисел по группам: (3, 8, 7), (2, 11, 9), (1, 4, 6).

В ответе укажите два числа: сначала искомое значение для файла A, затем для файла B.

Работаем с массивами, списками, прямыми и обратными остатками

Экзотические (дичайшие) задачи

27 (№ 4429) (Е. Драчева) Набор данных состоит из групп натуральных чисел, каждая группа записана в отдельной строке. В любой группе содержится не менее двух чисел. Из каждой группы выбрали два числа и нашли их наименьшее общее краткое (НОК). Затем все полученные таким образом значения НОК сложили. Определите наибольшую сумму, кратную числу 5 или 7 (но не одновременно двум этим числам), которая может быть получена таким образом.

Входные данные. Даны два входных файла (файл A и файл B), каждый из которых содержит в первой строке количество чисел N ($2 \le N \le 100000$). В каждой из следующих N строк файлов записан сначала размер группы K ($N <= 10$), а затем – K натуральных чисел, не превышающих 500.

Пример входного файла:

```
4
2 8 6
3 2 7 8
2 6 5
4 7 3 8 6
```

Для указанных входных данных значения НОК для первой группы – 24; для второй группы – 14, 8, 56; для третьей группы – 30, для четвёртой группы – 6, 21, 24, 24, 42, 56. Значением искомой суммы должно быть число 110 (24+14+30+42).

В ответе укажите два числа: сначала искомое значение для файла A, затем для файла B.

И остатки, и массивы, и множества, и словари, и списки, и строки, и комбинаторика, и математический анализ - используем абсолютно всё

Как ещё можно классифицировать 27-е задачи?

1) обработка одной последовательности (один ряд чисел)
2) обработка двойной последовательности (ряды пар чисел)
3) обработка тройной последовательности (ряды троек чисел)
4) обработка n-й последовательности (пока из рода “космическое”)

В зависимости от этого выполняется организация данных во входном файле (файле, в котором хранятся числовые последовательности, которые в ходе решения задачи нам предстоит обработать. В первом случае мы будем иметь дело лишь с одним элементом в строке, во втором случае - с двумя элементами, разделенными пробелом, при этом оба придется обработать - в этом заключена принципиальная разница подхода к решению

Зачем нужен ЯП?

Зачем же нужен ЯП при решении 27-х задач? Приведем банальный пример.

“Дана последовательность из 10 миллионов целых чисел. Найти количество нечетных чисел, кратных 17 и 24”. Очевидно, данная задача не решается вручную, но если вдруг пытаться = времязатратно, не исключены ошибки в силу невнимательности и усталости. Приведем решение задачи в Python 3:

```
with open('27.txt') as f: ФАЙЛ ОТКРЫТ
    n = int(f.readline()) КОЛ-ВО ЧИСЕЛ
    count = 0 СЧЕТЧИК
    for i in range(n): ЦИКЛ
1-е число - x = int(f.readline())    //  УСЛОВИЕ НА ВЕЛ.
        if x%2 != 0 and x%17 == 0 and x%24 == 0:
            count += 1
    print(count) ОТВЕТ
```

Каков необходимый минимум знаний и навыков?

- Базовый уровень владения ЯП >=: здесь без вариантов
- Теория чисел: остатки и кратность, математическая индукция
- Углубленно: массивы, списки, словари, функции и библиотеки
- Понимание простейших алгоритмов
- Различные методы решения задач

С чего следует начинать при подготовке?

Начать можно с 17-х задач нового формата, которые предполагают k проходов по последовательности целых чисел, где k >= 1: O(k*n). Можно сказать, что современная 17-я задача = “простая 27-я на минималках”. Далее следует попробовать себя в решении 26-х задач путем написания кода: практика в работе со списками и массивами - сортировка, упорядочивание, поиск максимума/минимума, сравнение, подобие задачи об NP-полной задаче комбинаторной оптимизации (о рюкзаке). После чего можно научиться писать переборное решение, дабы идея задачи всплыла на поверхность, если она не очевидна. Уже далее переходим к решению самых простых задач, с постепенным усложнением, “эволюционируем”

Предупреждение и наставление!

В разделах ПРАКТИКА по блокам я НЕ БУДУ давать подробное разъяснение, а задачи иногда будут чуть сложнее, нежели те, что разобраны в теоретической части. Почему? - ВАЖНО ДУМАТЬ И АНАЛИЗИРОВАТЬ: пытайтесь решить задачу самостоятельно, если не получилось ⇔ на следующем слайде имеется код с решением, но без разъяснений - пытайтесь понять и осознать суть решения самостоятельно, иначе и смысла не будет.

Ни в какую не получается - пишем автору:

VK - https://vk.com/leonid_shastin

Где брать файлы для практики?

В разделах ПРАКТИКА по блокам, если Ваше решение будет отличаться от нашего, авторского, еще не значит, что оно неправильное, ведь задачу можно решить различными методами. Поэтому, файлы для каждого задания Вы сможете найти по этой ссылке: https://disk.yandex.ru/d/Midnf646qx9vCw.

Если ответ сходится с ответом, которое получается в результате авторского решения - поздравляем! Номер блока и задания соответствует имени папки, в которой расположены нужные файлы.

Столкнулись с проблемами - пишем автору:

VK - https://vk.com/leonid_shastin

Знакомство #1: простейшие задачи

Решим легчайшие задачи, что будут являться некой подводкой к дальнейшему изучению реальных 27-х. Задачи, что будут представлены далее, крайне просты. Если их решение Вам непонятно - практикуйтесь в решении задач первой части, значит, не пришло еще время. После познакомимся с теорией чисел...

#1: №1

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти сумму максимального и минимального чисел, входящих в последовательность.

Приведем решение задачи в Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    INF = float('inf') #Абсолютно максимальное число
    maxi, mini = 0, INF
    for i in range(n):
        x = int(f.readline())
        maxi = max(maxi, x)
        mini = min(mini, x)
    print(sum({maxi, mini}))
```

#1: №2

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти сумму трех (различных по индексу) наименьших чисел, входящих в последовательность.

Приведем решение задачи в Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    INF = float('inf') #Абсолютно максимальное число
    k = [INF, INF, INF]
    for i in range(n):
        x = int(f.readline())
        k[0] = min(k[0], x)
        k.sort(reverse = True) #Сортировка по убыванию
    print(sum(k))
```

#1: №3

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти количество чисел, которые оканчиваются и начинаются на одну и ту же цифру.

Приведем решение задачи в Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    count = 0
    for i in range(n):
        x = int(f.readline())
        count += (str(x)[0] == str(x)[-1])
    print(count)
```

#1: №4

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Определить, на какую цифру чаще всего оканчиваются элементы в последовательности.

Приведем решение задачи в Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k = [0]*10
    high = [False]*2
    for i in range(n):
        x = int(f.readline())
        k[x%10] += 1
        if k[x%10] > high[0]:
            high[0] = k[x%10]
            high[1] = x%10
    print(high[1])
```

#1: №5

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Определить максимальную разность двух элементов последовательности.

Приведем решение задачи в Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    INF = float('inf')
    x0, x1 = 0, INF
    for i in range(n):
        x = int(f.readline())
        if x > x0: x0 = x
        elif x < x1: x1 = x
    print(x0 - x1)
```

Работа с делимостью #2: “прямые” остатки

В этом блоке разберёмся с “прямыми“ остатками, в будущем рассмотрим такую вещь как “обратные остатки”. Так что такое остаток от суммы (x, y) % z, если речь идет о делении суммы чисел (x, y) на z?

“Прямой” остаток ((x + y) / z) = (x / z + y / z) !

Каков остаток мы имеем при делении суммы чисел (5, 15) на 5:

(5 % 5 = 0; 15 % 5 = 0) ⇔ ((5 + 15) % 5 = 0 + 0 = 0)

А при делении суммы чисел (8, 16) на 7:

(8 % 7 = 1; 16 % 7 = 2) ⇔ ((8 + 16) % 7 = 2 + 1 = 3)

В рамках делимости, ((z1 + z2) = z) ⇔ ((z1 + z2) = 0) , где z1 - остаток от деления на z числа x, а z2 - остаток от деления на z числа y. Остатки двух чисел складываются по модулю, и если их сумма = z ⇔ 0, значит, два числа делятся на соответствующий z нацело без остатка

“Прямой” остаток ((z + z) / z) = (z / z + z / z) = 0 !

Каков остаток мы имеем при делении суммы чисел (5, 5) на 5:

(5 % 5 = 0; 5 % 5 = 0) ⇔ ((5 + 5) % 5 = 0 + 0 = 0)

А при делении суммы чисел (7, 7) на 7:

(7 % 7 = 0; 7 % 7 = 0) ⇔ ((7 + 7) % 7 = 0 + 0 = 0)

В рамках делимости, ((z + z) / z) ⇔ 0, где z - единое число, остатки двух равных чисел складываются по модулю, их сумма = z ⇔ 0, значит, два числа (z) делятся на себя же (z) нацело без остатка

“Прямой” остаток ((x * y) / z) = (x / z * y / z) !

Каков остаток мы имеем при делении произведения чисел (5, 15) на 5:

(5 % 5 = 0; 15 % 5 = 0) ⇔ ((5 * 15) % 5 = 0 * 0 = 0)

А при делении произведения чисел (8, 16) на 7:

(8 % 7 = 1; 16 % 7 = 2) ⇔ ((8 * 16) % 7 = 2 * 1 = 2)

В рамках делимости, ((z1 * z2) = z) ⇔ ((z1 * z2) = 0) , где z1 - остаток от деления на z числа x, а z2 - остаток от деления на z числа y, остатки двух чисел умножаются по модулю, и если их произведение = z ⇔ 0, значит, произведение двух чисел делится на соответствующий z нацело без остатка

“Прямой” остаток ((z * z) / z) = 0 !

Каков остаток мы имеем при делении произведения чисел (5, 5) на 5:

(5 % 5 = 0; 5 % 5 = 0) ⇔ ((5 * 5) % 5 = 0 * 0 = 0)

А при делении произведения чисел (7, 7) на 7:

(7 % 7 = 0; 7 % 7 = 0) ⇔ ((7 * 7) % 7 = 0 * 0 = 0)

В рамках делимости, ((z * z) / z) ⇔ 0, где z - единое число, остатки двух равных чисел умножаются по модулю, их произведение = z ⇔ 0, значит, два числа z делятся на себя же (z) нацело без остатка

“Прямой” остаток ((x * z) / z) = 0 !

Каков остаток мы имеем при делении произведения чисел (3, 5) на 5:

(3 % 5 = 3; 5 % 5 = 0) ⇔ ((3 * 5) % 5 = 3 * 0 = 0)

А при делении произведения чисел (3, 7) на 7:

(3 % 7 = 3; 7 % 7 = 0) ⇔ ((3 * 7) % 7 = 3 * 0 = 0)

В рамках делимости, ((x * z) / z) ⇔ 0, где z - единое число, остатки двух равных чисел умножаются по модулю, их произведение = z ⇔ 0, значит, два числа x и z делятся на соответствующий z нацело без остатка

Закрепим материал, ответив на вопросы...

- Какой остаток от деления на 3 получается в результате произведения чисел (796, 295432) ?
- Какой остаток от деления на 15 получается в результате произведения чисел (21371^5, 15) ?
- Какой остаток от деления на 6 получается в результате суммы чисел (611, 1719) ?
- Какой остаток от деления на 11 получается в результате суммы чисел (11, 11) ?

Ответы и решения

- (796 * 295432) % 3 ⇔ (796 % 3 = 1) * (295432 % 3 = 1) ⇔ 1 * 1 = 1
- (21371^5 * 15) % 15 ⇔ (21371^5 % 5 = y) * (15 % 5 = 0) ⇔ y * 0 = 0
- (611 + 1719) % 6 ⇔ (611 % 6 = 5) + (1719 % 6 = 3) ⇔ 5 + 3 = 8 > 6 ⇔ 8 - 6 = 2
- (11 + 11) % 11 ⇔ (11 % 11 = 0) + (11 % 11 = 0) ⇔ 0 + 0 = 0

Работа с делимостью #2: простейшие задачи

Решим легчайшие задачи, связанные с делимостью и остатками. Закрепим изученный материал на практике...

#2: №1

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти наибольшую сумму двух чисел, кратную 3.

Рассуждаем: нас интересует случай, при котором выполняется равенство остатков (x + y) % 3 = 0 ⇔ ((x % 3) + (y % 3)) = 0. В каких случаях равенство может выполняться?

- x % 3 = 0; y % 3 = 0 ⇔ 0 + 0 = 0
- x % 3 = 1; y % 3 = 2 ⇔ 1 + 2 = 3 ⇔ 3 - 3 = 0
- x % 3 = 2; y % 3 = 1 ⇔ 2 + 1 = 3 ⇔ 3 - 3 = 0

Решение задачи на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k0, k02 = 0, 0 #наибольшие числа, ост. от дел-я на 3 для которых = 0
    k1 = 0 #наибольшее число, остаток от деления на 3 для которого = 1
    k2 = 0 #наибольшее число, остаток от деления на 3 для которого = 2
    for i in range(n):
        x = int(f.readline())
        if x%3 == 1:
            k1 = max(k1, x) #находим макс. число с ост. = 1
        if x%3 == 2:
            k2 = max(k2, x) #находим макс. число с ост. = 2
        if x%3 == 0 and x > k0:
            k02 = k0
            k0 = x #находим макс. число с ост. = 0
        elif x%3 == 0 and x > k02:
            k02 = x #находим предмаксимальное число с ост. = 0
    print(max(k0 + k02, k1 + k2)) #выводим макс. сумму из возможных
```

#2: №2

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти наибольшее произведение двух чисел, кратное 3.

Рассуждаем: нас интересует случай, при котором выполняется равенство остатков (x * y) % 3 = 0 ⇔ ((x % 3) * (y % 3)) = 0. В каких случаях равенство может выполняться?

- x % 3 = 0; y % 3 = 0 ⇔ 0 * 0 = 0
- x % 3 = 0; y % 3 = z ⇔ 0 * z = 0; z - любое число
- x % 3 = z; y % 3 = 0 ⇔ z * 0 = 0; z - любое число

Решение задачи на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k0, k02 = 0, 0 #наибольшие числа, ост. от дел-я на 3 для которых = 0
    kmax = 0 #наибольшее число z с любым остатком от дел-я на 3
    for i in range(n):
        x = int(f.readline())
        if x%3 == 0 and x > k0:
            k02 = k0
            k0 = x #находим макс. число с ост. = 0
        elif x%3 == 0 and x > k02:
            k02 = x #находим предмаксимальное число с ост. = 0
        else:
            kmax = max(kmax, x) #находим макс. число с любым остатком
    print(max(k0*k02, kmax*k0)) #выводим макс. пр-е из возможных
```

#2: №3

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти наименьшее произведение трех чисел, кратное 7.

Рассуждаем: нас интересует случай, при котором выполняется равенство остатков (x * y * z) % 7 = 0 ⇔ ((x % 7) * (y % 7) * (z % 7)) = 0. В каких случаях равенство может выполняться?

- x % 7 = 0; y % 7 = 0; z % 7 = 0 ⇔ 0 * 0 * 0 = 0
- x % 7 = 0; y % 7 = t; z % 7 = t ⇔ 0 * z * z = 0; t - любое число
- x % 7 = t; y % 7 = 0; z % 7 = t ⇔ z * 0 * z = 0; t - любое число
- x % 7 = t; y % 7 = t; z % 7 = 0 ⇔ z * z * 0 = 0; t - любое число

- Еще какие-то комбинации? ⇔ Суть ясна, нужны тройки чисел, среди которых хотя бы одно (x or y or z) % 7 = 0

Решение задачи на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    INF = float('inf')  #Абсолютно большое число
    k0, k02, k03 = INF, INF, INF #переменные для наименьших чисел с ост. = 0
    kmin, kpredmin = INF, INF #переменные для наименьших чисел с любым ост.
    for i in range(n):
        x = int(f.readline())
        if x%7 == 0 and x < k0:
            k03 = k02
            k02 = k0
            k0 = x
        elif x%7 == 0 and x < k02:
            k03 = k02
            k02 = x
        elif x%7 == 0 and x < k03:
            k03 = x
        elif x%7 != 0 and x < kmin:
            kpredmin = kmin
            kmin = x
        elif x%7 != 0 and x < kpredmin:
            kpredmin = x
        print(min(k0*k02*k03, kmin*kpredmin*k0, k0*k02*kmin)) #возможные комбинации
```

Обработка остатков #3: массивы

В прошлом блоке разобрали базис “прямых” остатков. Как Вы поняли, существуют некие комбинации из остатков, которые могут в результате давать тот или ной остаток z путем конъюнкции и дизъюнкции. А теперь представим задачу: “Найти наибольшую сумму трех чисел, кратную k = 517”. Не заводить же нам несколько сотен переменных для решения этой задачи? Не писать же нам сотни условий? Для наиболее простой и удобной обработки остатков мы обратимся к массивам. В этом же блоке разберемся с задачами посложнее и поинтереснее =)

Чем нам поможет массив в решении задач?

Ранее было сказано, что массивы мы будем использовать для наиболее удобной и динамической обработки различных остатков (x % z), где x и z - целые числа. Как конкретно мы будем использовать массивы? Разберём на примере задачи №2 из прошлого блока. Вспоминаем идею задачи: нам нужно обрабатывать остатки от деления на z так, чтобы в результате найти наибольшее произведение двух чисел (x * y), кратное z. Реализуем же это!

Объявим массив ⇔

```
k = [0,0,0] ⇔  k = [0]*3
```

Имеем массив размерности = z = 3. Не забываем, что индексация (нумерация) в массиве ведется с нуля:

```
[0,0,0]
 0 1 2
```

Уже догадались, почему размерность необходимого нам массива = z (числу, на которое должно делиться искомое произведение)? В массиве в ячейке 0 мы будем хранить информацию об элементах с остатком % 3 = 0; в ячейке 1 с остатком % 3 = 1; в ячейке 2 с остатком % 3 = 2; соответственно

Проходясь по циклу, мы, с каждой итерацией, будем обновлять информацию, хранящуюся в массиве, так она будет постоянно актуальна

Как обновлять информацию в массиве?

Мы хотим найти наибольшее произведение двух элементов, кратное 3. Поэтому, в массиве будем хранить наибольшие числа с соответствующими остатками, которые удалось найти на настоящий момент времени. С каждой итерацией мы будем обновлять информацию в массиве, если это будет необходимо. Так это будет выглядеть:

```
k[x%3] = max(k[x%3], x)
```

Как мы в итоге получим ответ?

Продолжаем рассматривать на примере всё той же задачи:

```
for y in range(3):
    if (x*k[y])%3 == 0:
        maxi = max(maxi, x*k[y])
```

С каждой итерацией мы проходимся по z, в данном случае z = 3, проверяем, делится ли произведение (x * k[y]) на z; x - текущий элемент, k[y] - элемент в массиве с соответствующим остатком. Если делится - проверяем maxi (в maxi хранится наибольшее произведение) на максимум, если текущее произведение больше => обновляем информацию, хранящуюся в maxi

Рассмотрим полное решение на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k = [0]*3
    maxi = 0
    for i in range(n):
        x = int(f.readline())
        for y in range(3):
            if (x*k[y])%3 == 0:
                maxi = max(maxi, x*k[y])
        k[x%3] = max(k[x%3], x)
    print(maxi)
```

Обработка остатков #3: задачи средней сложности

Решим легкие и средние задачи, связанные с делимостью и остатками, с изученными блоками #2 и #3. Закрепим изученный материал на практике...

#3: №1

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти наибольшую сумму двух чисел, кратную 90.

Рассуждаем: нам нужно проанализировать числа со всевозможными остатками от деления на z = 90, их будет много, создадим массив, который далее будем обновлять с каждой итерацией:

- k = [0]*90

Решение задачи на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k = [0]*90
    maxi = 0
    for i in range(n):
        x = int(f.readline())
        for y in range(90):
            if (x+k[y])%90 == 0:
                maxi = max(maxi, x+k[y])
        k[x%90] = max(k[x%90], x)
    print(maxi)
```

#3: №2

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти наибольшее чётное произведение двух чисел, кратное 13.

Рассуждаем: нам нужно проанализировать числа со всевозможными остатками от деления на z1 = 13 и z2 = 2, их будет много, создадим массив, который далее будем обновлять с каждой итерацией на (z1*z2) = (13*2) = 26; здесь мы вспомнили блок #2:

- k = [0]*26

Решение задачи на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k = [0]*26
    maxi = 0
    for i in range(n):
        x = int(f.readline())
        for y in range(26):
            if (x*k[y])%26 == 0:
                maxi = max(maxi, x*k[y])
        k[x%26] = max(k[x%26], x)
    print(maxi)
```

#3: №3

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти количество пар чисел, сумма которых четна.

Рассуждаем: что-то новенькое ⇔ не стоит бояться новых формулировок ⇔ нам нужно проанализировать пары чисел со всевозможными остатками от деления на z = 2, найти количество таких, сумма которых четна. Вновь работаем с массивами, ничего сложного:

- k = [0]*2
- count = 0 ⇔ количество подходящих нам пар, счётчик

Решение задачи на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k = [0]*2 #храним количество чисел, а не макс. / мин.
    count = 0
    for i in range(n):
        x = int(f.readline())
        for y in range(2):
            if (x+y)%2 == 0: #если ((x % 2) + (y % 2) = 0)
                count += k[y] #наращиваем на кол-во таких пар
        k[x%2] += 1 #обновляем количество чисел с нужным остатком
    print(count)
```

Работа с делимостью #4: “дополняющие” остатки

В блоке #2 разобрались с “прямыми” остатками. В этом же блоке рассмотрим “дополняющие” ⇔ “обратные” остатки. Так что такое дополняющий остаток от (x, y) , если речь идет о делении суммы чисел (x, y) на z?

Как работают “дополняющие” остатки?

Предположим, есть ситуация, в ходе которой мы утверждаем, что выражение: (x + y) % z = 0. Преобразовывая выражение, согласно теории в блоке #2, мы получим: (x + y) % z = 0 ⇔ ((x % z) + (y % z) = 0). Пусть x = 17, а z = 5, чему тогда равен y? ⇔ ((x % 5 = 2) + (y % 5 = ?) = 0) ⇔ 2 + ост.(y) = 0. Вспоминаем, что в рамках делимости 0 = z ⇔ 0 = 5. Тогда: 2 + ост.(y) = 0 ⇔ 2 + ост.(y) = 5 ⇔ ост.(y) = 5 - 2 ⇔ ост.(y) = 3. То есть, чтобы текущая сумма (17 + y) была кратна 5, нужно, чтобы y имел остаток = 3, что мы доказали в ходе преобразований

Рассмотрим на других примерах

- Какой остаток от деления на 3 должен иметь y, чтобы (x + y) было кратно 3, ныне x = 14? Решаем: (x + y) % 3 = 0 ⇔ (x%3 + y%3) = 0 ⇔ (14%3 + y%3) = 0 ⇔ 2 + y%3 = 0 ⇔ 2 + ост.(y) = 0 = 3 ⇔ ост.(y) = 3 - 2 = 1.

- Какой остаток от деления на 7 должен иметь y, чтобы (x + y) было кратно 7, ныне x = 14? Решаем: (x + y) % 7 = 0 ⇔ (x%7 + y%7) = 0 ⇔ (14%7 + y%7) = 0 ⇔ 0 + y%7 = 0 ⇔ 0 + ост.(y) = 0 ⇔ ост.(y) = 0 - 0 = 0.

Формула “дополняющего” остатка

Выведем формулу дополняющего остатка.

Пусть интересующий остаток = ост.(y); число, которое нужно дополнить = x, тогда нас интересует остаток числа y от деления на z, итого получаем:

Ост.(y) = (z - x % z) % z - итоговая формула.

Например, найдем “дополняющий остаток” для (15 + y) % 6 = 0:

Ост.(y) = (z - x % z) % z ⇔ (6 - 15 % 6) % 6 ⇔ (6 - 3) % 6 = 3 % 6 = 3

Работа с делимостью #4: практика - задачи

Решим легкие и средние задачи, связанные с делимостью и остатками, с изученными блоками #2 - #4. Закрепим изученный материал на практике...

#4: №1

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти количество пар чисел, сумма которых кратна 17.

Рассуждаем: используем то, что изучили - массивы, “прямые” и “дополняющие” остатки:

- k = [0]*17
- count = 0 ⇔ количество подходящих нам пар, счётчик
- И формула “дополняющего остатка” ⇔ ost = (17 - x % 17) % 17

Решение задачи на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k = [0]*17
    count = 0
    for i in range(n):
        x = int(f.readline())
        ost = (17 - x%17)%17
        count += k[ost]
        k[x%17] += 1
    print(count)
```

#4: №2

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти пару чисел с наибольшей суммой, кратной 76.

Рассуждаем: используем то, что изучили - массивы, “прямые” и “дополняющие” остатки:

- k = [0]*76
- maxi = 0 ⇔ максимальная сумма
- И формула “дополняющего остатка” ⇔ ost = (76 - x % 76) % 76

Решение задачи на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k = [0]*76
    maxi = 0
    for i in range(n):
        x = int(f.readline())
        ost = (76 - x%76)%76
        maxi = max(maxi, x + k[ost]) if k[ost] != 0 else maxi
        k[x%76]  = max(k[x%76], x)
    print(maxi)
```

#4: №3

В файле ‘27.txt’ представлена последовательность N целых чисел. В первой строке находится число N, следом ровно N целых чисел. Найти количество пар чисел, сумма которых кратна 19, при этом хотя бы один элемент пары > 50.

Рассуждаем: используем то, что изучили - массивы, “прямые” и “дополняющие” остатки:

- k = [0]*19 количество всех чисел по остаткам
- count = 0 ⇔ количество подходящих пар; счётчик
- Формула “дополняющего остатка” ⇔ ost = (19 - x % 19) % 19
- k_big = [0]*19 ⇔ количество чисел, больших 50 по остаткам

⇔ Обработаем количество пар с помощью двух массивов с разными функциями

Решение задачи на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    k = [0]*19
    k_big = [0]*19
    count = 0
    for i in range(n):
        x = int(f.readline())
        ost = (19 - x%19)%19
        if x > 50: #если текущий элемент больше 50
            count += k[ost] #он образовывает пары с любыми числами
            k_big[x%19] += 1 #счётчик больших наращивается
        else: #иначе
            count += k_big[ost] #образовывает пары с большими числами
        k[x%19] += 1 #счётчик для всех чисел наращивается вне условия
    print(count)
```

Практика #5: решаем разные задачи

В блоках #1 - #4 разобрались с делимостью, различными видами остатков, массивами. В этом, #5 блоке, попрактикуемся в решении задач, которые теперь для нас доступны. Если возможно, рассмотрим разные варианты, как можно решить ту или иную задачу. Поехали же!

#5: №1

По каналу связи передавались данные в виде последовательности положительных целых чисел. Количество чисел заранее неизвестно, но не менее двух, признаком конца данных считается число 0. Контрольное значение равно такому максимально возможному произведению двух чисел из переданного набора, которое делится на 7, но не делится на 49. Если такое произведение получить нельзя, контрольное значение считается равным 1.

Программа должна напечатать одно число — вычисленное контрольное значение, соответствующую условиям задачи.

Пробуем решить! Подсказки:

- Признаком окончания ввода является 0 - обратите внимание
- k = [0]*49

Решение задачи на Python 3:

```
with open('27986_B.txt') as f:

    k  = [0]*49
    maxi = float('-inf')

    while True:
        x = int(f.readline())
        if x == 0: break

        for y in range(49):
            if (x*k[y])%7 == 0 and  (x*k[y])%49 != 0:
                maxi = max(maxi, x*k[y])

        k[x%7] = max(x, k[x%7])

    print(maxi)
```

Открываем файл, задаем массив размерности z = 49, переменную maxi, в ней хранится максимальное произведение. Пока не нашёлся 0 считываем x, если нашёлся 0, делаем брэйк (выходим из цикла). Проходимся по остаткам z, если текущее число при произведении на число с остатком y кратно 7 и не кратно 49, то перезаписываем maxi в случае необходимости (если maxi меньше текущего произведения), при этом на каждой итерации по циклу мы обновляем k[x%7] от остатка при делении на 7 на максимальное число, если это возможно. Выводим maxi ⇔ получаем ответ

#5: №2

На вход программы поступает последовательность из N целых положительных чисел. Рассматриваются все пары различных элементов последовательности (элементы пары не обязаны стоять в последовательности рядом, порядок элементов в паре не важен). Необходимо определить количество пар, для которых произведение элементов делится на 26.

В первой строке входных данных задаётся количество чисел N ($1 \le N \le 60000$). В каждой из последующих N строк записано одно целое положительное число, не превышающее 10 000. В качестве результата программа должна напечатать одно число: количество пар, в которых произведение элементов кратно 26.

Пробуем решить! Подсказки:

- k = [0]*26; храним количество чисел по остаткам, а не максимумы
- count = 0; счетчик чисел

Решение задачи на Python 3:

```
with open('27989_B.txt') as f:
    n = int(f.readline())

    k  = [0]*26
    count = 0

    for i in range(n):
        x = int(f.readline())

        for y in range(26):
            if (x*y)%26 == 0:
                count += k[y]

        k[x%26] += 1

    print(count)
```

Открываем файл, задаем массив размерности z = 26, переменную count, в ней хранится количество подходящих нам пар. Проходимся по n (количеству чисел), считываем каждое новое число. Проходимся по остаткам z, если текущее число при произведении на число с остатком y кратно 26, то увеличиваем count на количество подходящих нам пар k[y], при этом на каждой итерации по циклу мы обновляем k[x%26] от остатка при делении на 26 (нашлось такое число, увеличиваем счётчик сохраненных). Выводим count ⇔ получаем ответ

#5: №3

Дана последовательность N целых положительных чисел. Необходимо определить количество пар элементов этой последовательности, сумма которых делится на $m = 80$ и при этом хотя бы один элемент из пары больше $b = 50$.

Пробуем решить! Подсказки:

- k = [0]*80; храним количество всех чисел пар по остаткам
- count = 0; счетчик чисел
- big_k = [0]*80; храним количество чисел, больших 50 по остаткам

Решение задачи на Python 3:

```
with open('28130_B.txt') as f:
    n = int(f.readline())

    k = [0]*80
    big_k = [0]*80
    count = 0

    for i in range(n):
        x = int(f.readline())

        for y in range(80):
            if x > 50:
                if (x + y)%80 == 0:
                    count += k[y]
            else:
                if (x + y)%80 == 0:
                    count += big_k[y]

        if x > 50:
            big_k[x%80] += 1

        k[x%80] += 1

    print(count)
```

Открываем файл, задаем массив размерности z = 80, переменную count, в ней хранится количество подходящих нам пар, второй массив для чисел, больших 50, размерности тоже z = 80. Проходимся по n (количеству чисел), считываем каждое новое число. Проходимся по остаткам z, если текущее число при сумме с остатком y кратно 2, то если число больше 50, тогда увеличиваем count на количество подходящих нам пар k[y], иначе увеличиваем count на количество подходящих пар big_k[y], при этом на каждой итерации по циклу мы обновляем k[x%80] от остатка при делении на 80 (нашлось такое число, увеличиваем счётчик сохраненных), а если число больше 50, то обновляем и количество big_k[x%80].Выводим count ⇔ получаем ответ

#5: №4

Последовательность натуральных чисел характеризуется числом X — наибольшим числом, кратным 14 и являющимся произведением двух элементов последовательности с различными номерами. Гарантируется, что хотя бы одно такое произведение в последовательности есть.

Пробуем решить! Подсказки:

- k = [0]*14; храним наибольшие числа по остаткам
- maxi = 0; максимальное произведение

Решение задачи на Python 3:

```
with open('27-B_2.txt') as f:
    n = int(f.readline())

    k = [0]*14
    maxi = float('-inf')

    for i in range(n):
        x = int(f.readline())

        for y in range(14):
            if (x*k[y])%14 == 0:
                maxi = max(maxi, x*k[y])

        k[x%14] = max(k[x%14], x)

    print(maxi)
```

Открываем файл, задаем массив размерности z = 14, переменную maxi, в ней хранится максимальное произведение. Проходимся по n (количеству чисел), считываем каждое новое число. Проходимся по остаткам z, если текущее число при произведении на число k[y] дает результат, кратный 14, то обновляем maxi (в том случае, если maxi меньше текущего произведения). При этом на каждой итерации не забываем обновлять k[x%14] на максимум, если это возможно. Выводим maxi ⇔ получаем ответ

#5: №5

New! Имеется набор данных, состоящий из положительных целых чисел, каждое из которых не превышает 1000. Они представляют собой результаты измерений, выполняемых прибором с интервалом 1 минута. Требуется найти для этой последовательности контрольное значение – наименьшую сумму квадратов двух результатов измерений, выполненных с интервалом не менее, чем в 5 минут.

Что-то новенькое… Нужно учитывать расстояние (интервал не менее чем в 5 минут), это же означает, что | i - j | >= 5, где i и j - различные индексы двух элементов последовательности. Как решать? Как учитывать расстояние?

Разберемся в следующем разделе #6 - задачи на расстояние!

Расстояние #6: работа с “буфером”

В предыдущем, #5 блоке, встретились с новым типом задач, которые предполагают учитывать расстояние между индексами различных элементов. В этом, #6 разделе, разберемся как решать задачи на расстояние методом “буфера”...

Какой еще “буфер” ?

Вы, возможно, знаете, что такое буфер обмена, - промежуточное хранилище данных, которое имеется как на ПК, так и на смартфонах. Копируете какой-то текст, например, информация отправляется во временной буфер. В Windows 10 даже присутствует такая вещь как “Журнал буфера обмена”, который позволяет удобно использовать буферизованную ранее информацию. Например, вы 10 раз скопировали различные элементы, с помощью журнала можно вернуться к каждому из них, удобно их обработать (вставить куда-то, предположим). При решении задач будем руководствоваться этой же логикой: зададим буфер (массив), который будем на ходу обрабатывать, при этом всё время хранить в нём какие-то элементы, что нужны нам будут в рамках той или иной задачи. Как это будет выглядеть?

Создаем так называемый буфер

Задача: найти наименьшую сумму квадратов чисел, расположенных на расстоянии не менее 5 ⇔ | i - j | >= 5.

Занесем данные в буфер для учета расстояния:

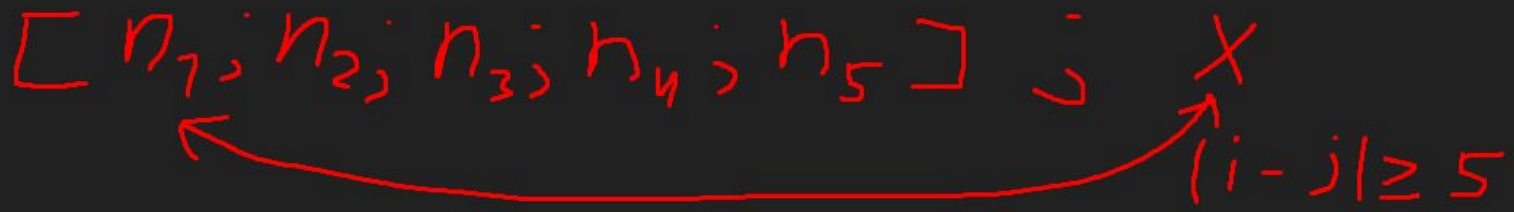
r = 5; расстояние

k = [int(f.readline()) for j in range(5)]; считали элементы с файла для r = 5

Теперь в буфере хранится 5 элементов. Далее, проходясь по оставшимся числам в цикле, не забываем учитывать, что n (количество чисел в файле, которые необходимо прочитать) уменьшилось на r = 5, поэтому, проходимся так: for i in range(n - r) ⇔ for i in range(n - 5)

Обрабатываем буферную информацию

Как теперь обработать буфер в соответствии с условием задачи?



Проходясь по циклу, если квадрат текущего числа x, которое мы прочитали, в сумме с квадратом числа n1 (для них | i - j | >= 5 ⇔ True) дает результат, меньший чем mini (mini объявили за минимальную сумму, дали ей значение абсолютного максимума ⇔ mini = float('inf')), в этом случае перезаписываем mini

Обновляем буферную информацию

Теперь нужно обновить наш буфер, добавив в него новое число, которое мы считали. Это мы будем делать с каждой итерацией. Одновременно с этим будем удалять лишний элемент, дабы размерность буфера оставалась = r, в рамках нашей задачи r = 5. Удалять мы будем самый бесполезный для нас элемент, дабы случайно не потерять ответ. В рамках задачи нужно найти наименьшую сумму, поэтому, мы будем удалять наибольший элемент:

k.append(x); добавили новое число в буфер

k.remove(max(k[0], k[1])); удаляем самый бесполезный элемент

Почему мы удаляем либо k[0], либо k[1]? Дабы учитывать расстояние между элементами: | i - j | >= 5

Рассмотрим полное решение на Python 3:

```
with open('27.txt') as f:
    n = int(f.readline())
    r = 5
    k = [int(f.readline()) for j in range(r)]
    mini = float('inf')
    for i in range(n - r):
        x = int(f.readline())
        if (x**2 + k[0]**2) < mini:
            mini = (x**2 + k[0]**2)
        k.append(x)
        k.remove(max(k[0], k[1]))
    print(mini)
```

Расстояние #6: практика - задачи

Решим легкие и средние задачи, связанные с делимостью, остатками и расстоянием, с изученными блоками #2 - #6. Закрепим изученный материал на практике...

#6: №1

(№ 2669) На спутнике «Восход» установлен прибор, предназначенный для измерения солнечной активности. Каждую минуту прибор передаёт по каналу связи неотрицательное целое число – количество энергии солнечного излучения, полученной за последнюю минуту, измеренное в условных единицах. Временем, в течение которого происходит передача, можно пренебречь. Необходимо найти в заданной серии показаний прибора минимальное нечётное произведение двух показаний, между моментами передачи которых прошло не менее 6 минут. Если получить такое произведение не удаётся, ответ считается равным «–1».

Рассуждаем:

Буферизуем: r = 6 ⇔ k = [int(f.readline()) for j in range(r)]

Не забываем: “если получить произведение не удаётся”, ответ считается равным “-1”

Решение задачи на Python 3:

```
with open('27-9b (1).txt') as f:
    n = int(f.readline())
    r = 6
    k = [int(f.readline()) for x in range(r)]
    mini = float('inf')
    for i in range(n - r):
        x = int(f.readline())
        if (x*k[0])%2 != 0: #если пр-е нечётное
            mini = min(mini, x*k[0])
        k.append(x)
        k.remove(max(k[0], k[1]))
    print(mini) if mini != float('inf') else print('-1')
```

#6: №2

(№ 2675) Имеется набор данных, состоящий из положительных целых чисел. Необходимо определить количество пар элементов (a_i, a_j) этого набора, в которых $1 \le \mathbf{i+5} \le j \le N$ и сумма элементов кратна 14.

Используем всё, чему научились:

Буферизуем: r = 5 ⇔ read = [int(f.readline()) for j in range(5)]

k = [0]*14; массив по остаткам от деления на 14

count = 0; счётчик подходящих пар

Для решения задачи заставим массивы “подружиться” =)

Решение задачи на Python 3:

```
with open('27-15b.txt') as f:
    n = int(f.readline())

    k = [0]*14
    count = 0
    read = [int(f.readline()) for i in range(5)]

    for i in range(n - 5):
        x = int(f.readline())
        k[read[0]%14] += 1
        count += k[(14 - x%14)%14]
        read.append(x)
        read.remove(read[0])

    print(count)
```

Открываем файл, задаем массив размерности z = 14, переменную count, в ней будем хранить количество подходящих пар. Буфер объявим = read на расстоянии r = 5. Проходимся по (n - r) ⇔ (n - 5), считываем каждое число. Увеличиваем количество чисел с остатком от деления на 14 в k[x%14] ⇔ k[read[0]%14], здесь в качестве x выступает read[0] (то число, которое находится на учтенном расстоянии r = 5). После увеличиваем count на количество чисел с “дополняющим” для числа x остатком, вспоминаем #4 блок. Не забываем обновлять и обрабатывать буфер - добавляем новый элемент и удаляем старый. Выводим count ⇔ получаем ответ

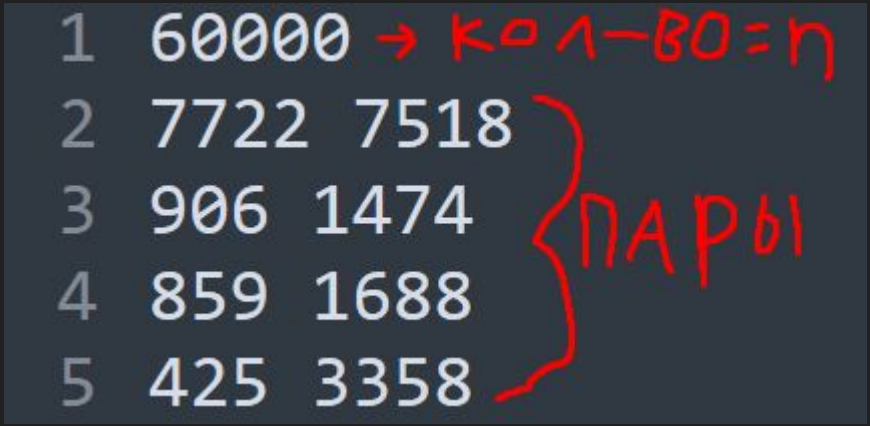
Пары чисел #7: метод “минимальной разности”

Начинаем разбирать задачи на пары чисел (два ряда чисел в файле). Суть в них состоит в поиске максимальной или минимальной суммы, кратной какому-то z, целому числу. Познакомимся с методом “минимальной разности”, а после будем переходить к МЧС - методу “частичных сумм”, рассматривать задачи как на пары чисел, так и на тройки (три ряда чисел в файле)...

Рассмотрим задачу...

(№ 2660) (Демовариант 2021 г.). Имеется набор данных, состоящий из пар положительных целых чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма всех выбранных чисел не делилась на 3 и при этом была максимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимально возможную сумму, соответствующую условиям задачи.

Что в файлике? ⇔



Пробуем решить

Как прочитать пару чисел в строке? (вдруг кто забыл):

```
a, b = map(int, f.readline().split())
```

Здесь split() ⇔ разделение по пробелам; map ⇔ обертка читаемых чисел в функцию int() ⇔ преобразование к целым, ведь изначально они определены как две строки. В a будем иметь число, стоящее слева, в b число, стоящее справа, соответственно.

Как получить максимальную сумму (с учетом того, что из каждой пары нужно выбрать только одно число)?

Получаем максимальную сумму

Чтобы получить наибольшую сумму, выбрав из каждой пары ровно одно число, будем выбирать наибольшее число из двух возможных на каждой итерации:

```
s += max(a, b) #s - итоговая максимальная сумма чисел
```

В итоге мы получаем наибольшую сумму, какую только возможно получить путем выбора одного из чисел в паре, но вот незадача - сумма делится на 3! В условии сказано, что итоговая сумма не должна делиться на 3, значит, нужно преобразовать сумму, заменив в какой-то паре больший элемент на меньший так, чтобы остаток от деления на 3 у суммы изменился, но при этом она все еще оставалась наибольшей, как мы это сделаем?

Подгоняем сумму до ответа

На помощь придет метод минимальной разности! Как мы уже выяснили, в какой-то паре нужно больший элемент заменить на меньший (к сумме прибавить меньший, а больший вычесть) ⇔ вычесть из суммы минимальную разность большего и меньшего элементов в какой-то паре, но так, чтобы остаток от деления на 3 у этой суммы изменился. Сейчас он равен нулю, т.е., сумма кратна 3. Как изменить кратность суммы: вычесть из нее число, которое имеет любой остаток от деления на 3, принадлежащий отрезку: [1, 2]. Если он будет иметь остаток 0, тогда 0 - 0 = 0, сумма останется кратной 3.

Находим минимальную разность

Создаем массив минимальных разностей по остаткам:

```
INF = float('inf')
min_razn = [INF, INF, INF]
```

На каждом шаге находим разность, если она не делится на 3, тогда обновляем массив min_razn с последствующим уменьшением, если это возможно:

```
cur_razn = abs(a - b) #текущая разность по модулю (>= 0)
if cur_razn%3:
    min_razn[cur_razn%3] = min(min_razn[cur_razn%3], cur_razn)
```

Получаем ответ ⇔ ура!

Остается вычесть из итоговой суммы минимальную разность, как следствие, мы получим максимальную сумму, которая не делится нацело на 3, что является, непосредственно, решением задачи:

```
print(s - min(min_razn))
```

Приведем полный код решения задачи:

```
with open('27-b (1).txt') as f:
    n = int(f.readline())
    INF = float('inf')
    min_razn = [INF, INF, INF]
    s = 0
    for i in range(n):
        a, b = map(int, f.readline().split())
        cur_razn = abs(a - b) #текущая разность по модулю (>= 0)
        if cur_razn%3:
            min_razn[cur_razn%3] = min(min_razn[cur_razn%3], cur_razn)
        s += max(a, b) #s - итоговая максимальная сумма чисел
    print(s - min(min_razn))
```

Чем плох метод “минимальной разности”?

Если Вы поняли идею, значит, врубаетесь: мы меняем итоговую сумму, прибавляя к ней или вычитая из нее минимальную разность. Но, что, если среди пар чисел нет чисел с нужным нам остатком? Тогда, очевидно, мы не сможем решить задачу, вычитая только одну разность ⇔ нужно сохранять множество минимальных разностей, комбинировать их друг с другом до тех пор, пока их сумма не даст число с необходимым для нас остатком. Здесь задача сводится уже к неподходящему для нас по времени решению, поскольку нам придется перебирать n разностей между собой n раз, при этом в комбинациях может быть k сумм. Отсюда вывод: лучше не использовать данный метод, но уметь и знать определенно не будет лишним...

Пары чисел #7: решим задачу для закрепления темы

(№ 2661) Имеется набор данных, состоящий из пар положительных целых чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма всех выбранных чисел не делилась на 3 и при этом была минимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – минимально возможную сумму, соответствующую условиям задачи.

Используем, что знаем:

k = [float('inf')]*3 ⇔ массив минимальных разностей

к итоговой сумме прибавим минимальный элемент из массива, она станет чуть больше, но при этом не будет делиться на 3;

не забываем учитывать, что число, которое будем прибавлять, должно иметь другой остаток от деления на 3, в отличие от

суммы ⇔ это мы учтем с помощью генератора непосредственно уже в момент вывода итогового ответа

Решение задачи на Python 3:

```
with open('27-1b.txt') as f:
    n = int(f.readline())

    k = [float('inf')]*3
    s = 0

    for _ in range(n):
        x, y = list(map(int, f.readline().split()))
        s += min(x, y)
        r = abs(x - y)
        k[r%3] = min(k[r%3], r)

    print(s) if s%3 != 0 else print(s + min({x for x in k if s%3 != x%3}))
```

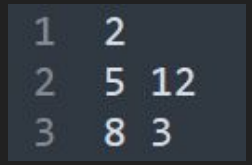
Пары чисел #8: метод “частичных сумм”

Начинаем разбирать задачи на пары чисел (два ряда чисел в файле). Суть в них состоит в поиске максимальной или минимальной суммы, кратной какому-то z, целому числу. Пришло время МЧС! В этом блоке рассмотрим, как он работает для пар чисел, в следующем же блоке решим задачи на тройки чисел с использованием этого же метода...

В чем состоит суть МЧС?

Суть метода заключается в накоплении максимальных (или минимальных, в зависимости от условия задачи), сумм с определенными остатками от деления на число z, которому должна быть кратна (или не кратна, в зависимости от условия задачи) итоговая сумма. Эти суммы мы будем обновлять на каждой итерации по циклу. Прикол заключается в том, что нам не нужно хранить абсолютно все суммы: нам нужно хранить z сумм, где z - число определения кратности, при этом эти суммы, в свою очередь, должны быть максимальными (или минимальными, в зависимости от условия задачи). Т. е., на каждом этапе решения мы будем рассматривать числа (пары, тройки, группы различной длины), которые дают остатки от 0 до z - 1

Рассмотрим МЧС на примере:

Дан ряд из n = 2 пар чисел: 

Необходимо выбрать из каждой пары ровно одно число, чтобы итоговая сумма не делилась на 3 и при этом была максимально возможной. s = [0] ⇔ массив (буфер) для сохранения частичных сумм, которые с каждой итерацией будут накапливаться.

Для первой пары: (5%3 = 2; 12%3 = 0) ⇔ s[0] = 12; s[1] = 0; s[2] = 5;

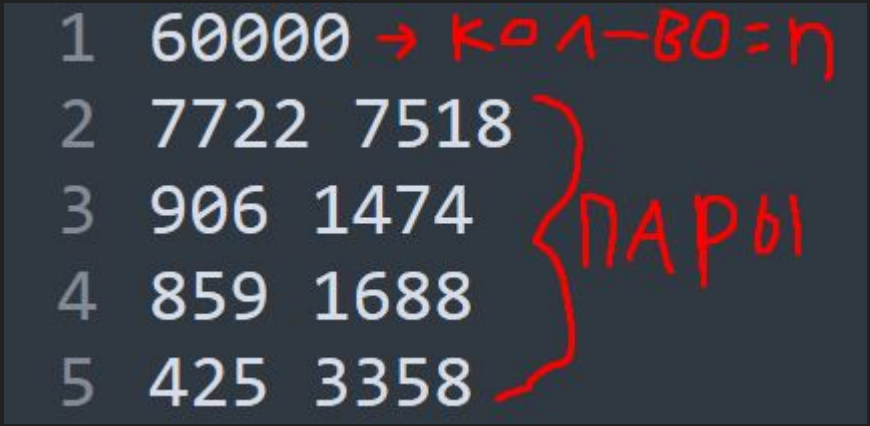
Для второй пары: (8%3 = 2; 3%3 = 0) ⇔ max(s[0]) = 12 + 3 = 15; max(s[1]) = 0 + 8 + 5 = 13; max(s[2]) = 12 + 8 = 20.

На этом шаге мы рассмотрели разные комбинации, выбрали из них те, что имеют наибольшую сумму и соответствующий остаток. В результате имеем ответ 20 - число, не кратное 3.

Рассмотрим задачу...

(№ 2660) (Демовариант 2021 г.). Имеется набор данных, состоящий из пар положительных целых чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма всех выбранных чисел не делилась на 3 и при этом была максимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимально возможную сумму, соответствующую условиям задачи.

Что в файлике? ⇔



Пишем код для МЧС...

Создадим буфер накопленных сумм ⇔ s = [0]

for i in range(n): ⇔ проходимся по циклу, далее…

pair = list(map(int, f.readline().split())) ⇔ прочитали пару

combinations = [a + b for a in s for b in pair] ⇔ образовали всевозможные комбинации пар, доступные нам вместе с уже сохраненными суммами в s и суммами в pair; далее создаем местный буфер:

s1 = [0]*3; после чего проходимся по combinations:

for x in combinations:

s1 = max(s1[x%3], x) ⇔ перезаписываем в s1 максимальную сумму с соответствующим остатком от деления на 3;

s = [x for x in s if x != 0] ⇔ обновляем основной буфер, если сумма != 0.

Получаем результат

В итоге в буфере s хранятся наибольшие суммы, которые удалось образовать, с соответствующими остатками. Выбираем сумму с тем остатком, который нас интересует, после чего выводим результат:

```
print(max(x for x in s if x%3 != 0))
```

Рассмотрим полное решение на Python 3:

```
with open('27-b (1).txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        pair = list(map(int, f.readline().split()))
        combinations = [a + b for a in s for b in pair]
        s1 = [0]*3
        for x in combinations:
            s1[x%3] = max(s1[x%3], x)
        s = [x for x in s1 if x != 0]
    print(max(x for x in s if x%3 != 0))
```

Пары чисел #8: практика - задачи

Решим легкие и средние задачи, связанные с МЧС (методом “частичных сумм”). Закрепим изученный материал на практике...

#8: №1

(№ 2664) Имеется набор данных, состоящий из пар положительных целых чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма всех выбранных чисел делилась на 5 и при этом была максимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимально возможную сумму, соответствующую условиям задачи.

Создаем буфер ⇔ s = [0]; далее по классике - типичный метод частичных сумм. На выходе нас интересует сумма, которая кратна 5, поэтому, будем выводить:

```
print(max(x for x in s if x%5 == 0))
```

Решение задачи на Python 3:

```
with open('27-4b.txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        pair = list(map(int, f.readline().split()))
        combinations = [a + b for a in s for b in pair]
        s1 = [0]*5
        for x in combinations:
            s1[x%5] = max(s1[x%5], x)
        s = [x for x in s1 if x != 0]
    print(max(x for x in s if x%5 == 0))
```

#8: №2

(№ 2681) Имеется набор данных, состоящий из пар положительных целых чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма всех выбранных чисел оканчивалась на 8 и при этом была максимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимально возможную сумму, соответствующую условиям задачи.

Создаем буфер ⇔ s = [0]; далее по классике - типичный метод частичных сумм. На выходе нас интересует сумма, которая оканчивается на 8, поэтому, будем выводить:

```
print(max(x for x in s if x%10 == 8))
```

Решение задачи на Python 3:

```
with open('27-21b.txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        pair = list(map(int, f.readline().split()))
        combinations = [a + b for a in s for b in pair]
        s1 = [0]*10 #размерность = 10, т.к. вариантов,
        #на что будет оканчиваться сумма ровно 10(0...9)
        for x in combinations:
            s1[x%10] = max(s1[x%10], x)
        s = [x for x in s1 if x != 0]
    print(max(x for x in s if x%10 == 8))
```

#8: №3

(№ 2684) Имеется набор данных, состоящий из пар положительных целых чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма всех выбранных чисел НЕ оканчивалась на 6 и при этом была минимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – минимально возможную сумму, соответствующую условиям задачи.

Создаем буфер ⇔ s = [0]; далее по классике - типичный метод частичных сумм. На выходе нас интересует сумма, которая не оканчивается на 6, поэтому, будем выводить:

```
print(max(x for x in s if x%10 != 6))
```

Решение задачи на Python 3:

```
with open('27-24b.txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        pair = list(map(int, f.readline().split()))
        combinations = [a + b for a in s for b in pair]
        #s1 наполним огромными числами, ведь ищем минимум
        s1 = [10**30]*10 #размерность = 10, т.к. вариантов,
        #на что будет оканчиваться сумма ровно 10(0...9)
        for x in combinations:
            s1[x%10] = min(s1[x%10], x) #берем минимум, поскольку
            #нас интересует минимально возможная сумма
        s = [x for x in s1 if x != 0]
    print(min(x for x in s if x%10 != 6))
```

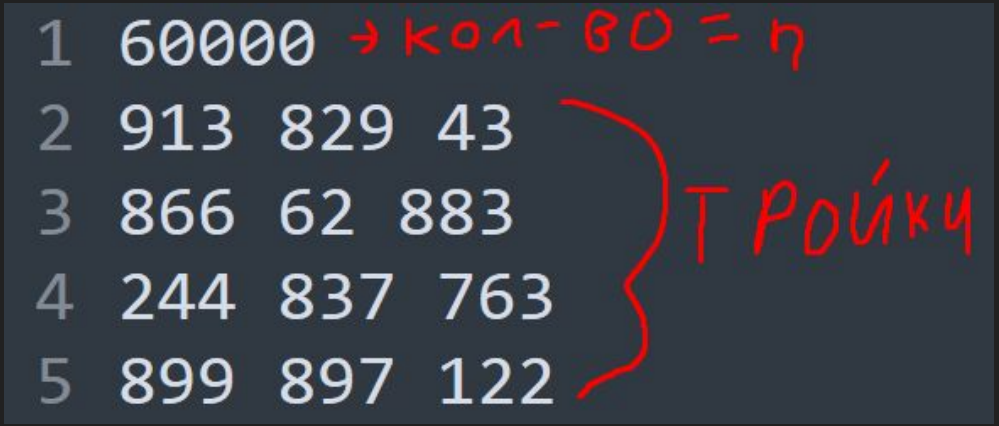
Тройки чисел #9: метод “частичных сумм”

Начинаем разбирать задачи на тройки чисел (три ряда чисел в файле). Суть в них состоит в поиске максимальной или минимальной суммы, кратной какому-то z, целому числу. Пришло время МЧС! В этом блоке рассмотрим, как он работает для троек чисел, в следующем же блоке усовершенствуем МЧС, упростим его для комфортной работы...

Рассмотрим задачу...

(№ 2690) Имеется набор данных, состоящий из троек положительных целых чисел. Необходимо выбрать из каждой тройки ровно одно число так, чтобы сумма всех выбранных чисел не делилась на 7 и при этом была минимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – минимально возможную сумму, соответствующую условиям задачи.

Что в файлике? ⇔



Пишем код для МЧС...

Создадим буфер накопленных сумм ⇔ s = [0]

for i in range(n): ⇔ проходимся по циклу, далее…

troyka = list(map(int, f.readline().split())) ⇔ прочитали тройку

combinations = [a + b for a in s for b in troyka] ⇔ образовали всевозможные комбинации, доступные нам вместе с уже сохраненными суммами в s и суммами в troyka; далее создаем местный буфер:

s1 = [10**30]*7; после чего проходимся по combinations:

for x in combinations:

s1 = min(s1[x%7], x) ⇔ перезаписываем в s1 минимальную сумму с соответствующим остатком от деления на 7;

s = [x for x in s if x != 0] ⇔ обновляем основной буфер, если сумма != 0.

Получаем результат

В итоге в буфере s хранятся наибольшие суммы, которые удалось образовать, с соответствующими остатками. Выбираем сумму с тем остатком, который нас интересует, после чего выводим результат:

```
print(min(x for x in s if x%7 != 0))
```

Как мы можем заметить, не поменялось абсолютно ничего. Почему? ⇔ Нужно выбирать из каждой тройки ровно одно число, та же ситуация была и с парами. Давайте рассмотрим другие задачи на тройки чисел: нужно будет выбирать два числа из каждой тройки, комбинаций становится больше...

Рассмотрим полное решение на Python 3:

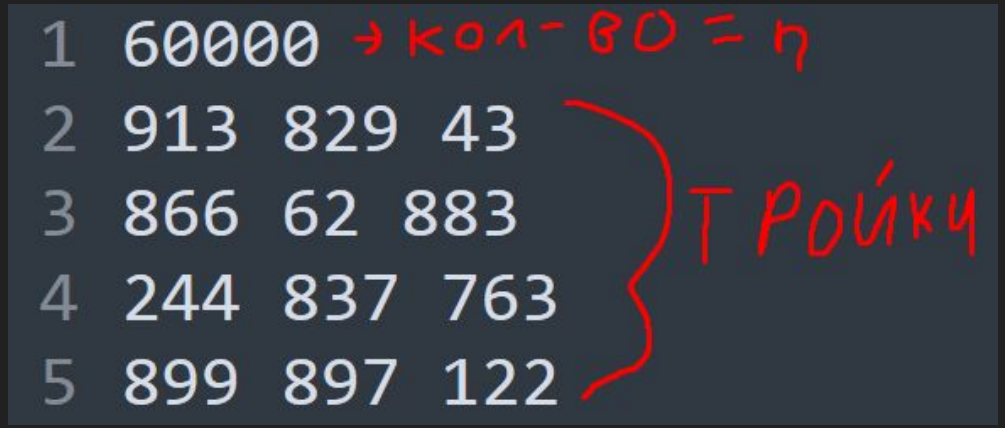
```
with open('27-30b.txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        troyka = list(map(int, f.readline().split()))
        combinations = [a + b for a in s for b in troyka]
        #s1 наполним огромными числами, ведь ищем минимум
        s1 = [10**30]*7

        for x in combinations:
            s1[x%7] = min(s1[x%7], x) #берем минимум, поскольку
            #нас интересует минимально возможная сумма
        s = [x for x in s1 if x != 0]
    print(min(x for x in s if x%7 != 0))
```

Рассмотрим задачу посложнее...

(№ 2689) Имеется набор данных, состоящий из троек положительных целых чисел. Необходимо выбрать из каждой тройки **два числа** так, чтобы сумма всех выбранных чисел не делилась на 5 и при этом была максимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимально возможную сумму, соответствующую условиям задачи.

Что в файлике? ⇔



Пишем код для МЧС...

Создадим буфер накопленных сумм ⇔ s = [0]

for i in range(n): ⇔ проходимся по циклу, далее…

a, b, c = list(map(int, f.readline().split())) ⇔ прочитали тройку

pairs = [a + b, a + c, b + c] ⇔ образовали комбинации по два выбранных числа

combinations = [a + b for a in s for b in pair] ⇔ образовали всевозможные комбинации, доступные нам вместе с уже сохраненными суммами в s и суммами в pair; далее создаем местный буфер:

s1 = [0]*5; после чего проходимся по combinations:

for x in combinations:

s1 = max(s1[x%5], x) ⇔ перезаписываем в s1 максимальную сумму с соответствующим остатком от деления на 5;

s = [x for x in s if x != 0] ⇔ обновляем основной буфер, если сумма != 0.

Получаем результат

В итоге в буфере s хранятся наибольшие суммы, которые удалось образовать, с соответствующими остатками. Выбираем сумму с тем остатком, который нас интересует, после чего выводим результат:

```
print(max(x for x in s if x%5 != 0))
```

Как мы можем заметить, решение практически не поменялось, мы лишь дополнительно сгенерировали всевозможные комбинации двух выбранных чисел в тройках (1-е с 3-м, 1-е со 2-м и 2-е с 3-м)

Рассмотрим полное решение на Python 3:

```
with open('27-29b.txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        a, b, c = list(map(int, f.readline().split()))
        pairs = [a + b, a + c, b + c]
        combinations = [a + b for a in s for b in pairs]
        #s1 наполним маленькими числами, ведь ищем максимум
        s1 = [0]*5
        for x in combinations:
            s1[x%5] = max(s1[x%5], x) #берем максимум, поскольку
            #нас интересует минимально возможная сумма
        s = [x for x in s1 if x != 0]
    print(max(x for x in s if x%5 != 0))
```

Тройки чисел #9: практика - задачи

Решим легкие и средние задачи, связанные с МЧС (методом “частичных сумм”). Закрепим изученный материал на практике...

#9: №1

(№ 2692) Имеется набор данных, состоящий из троек положительных целых чисел. Необходимо выбрать из каждой тройки ровно одно число так, чтобы сумма всех выбранных чисел делилась на 11 и при этом была минимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – минимально возможную сумму, соответствующую условиям задачи.

Создаем буфер ⇔ s = [0]; далее по классике - типичный метод частичных сумм. На выходе нас интересует сумма, которая кратна 11, поэтому, будем выводить:

```
print(min(x for x in s if x%11 == 0))
```

Решение задачи на Python 3:

```
with open('27-32b.txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        troyka = list(map(int, f.readline().split()))
        combinations = [a + b for a in s for b in troyka]
        #s1 наполним большими числами, ведь ищем минимум
        s1 = [10**30]*11
        for x in combinations:
            s1[x%11] = min(s1[x%11], x) #берем минимум, поскольку
            #нас интересует минимально возможная сумма
        s = [x for x in s1 if x != 0]
    print(min(x for x in s if x%11 == 0))
```

#9: №2

(№ 2691) Имеется набор данных, состоящий из троек положительных целых чисел. Необходимо выбрать из каждой тройки **два числа** так, чтобы сумма всех выбранных чисел не делилась на 9 и при этом была минимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – минимально возможную сумму, соответствующую условиям задачи.

Создаем буфер ⇔ s = [0]; далее по классике - типичный метод частичных сумм. На выходе нас интересует сумма, которая не кратна 9, поэтому, будем выводить:

```
print(min(x for x in s if x%9 != 0))
```

Решение задачи на Python 3:

```
with open('27-31b (2).txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        a, b, c = list(map(int, f.readline().split()))
        pairs = [a + b, a + c, b + c] #скомбинировали суммы по 2 числа
        combinations = [a + b for a in s for b in pairs]
        #s1 наполним большими числами, ведь ищем минимум
        s1 = [10**30]*9
        for x in combinations:
            s1[x%9] = min(s1[x%9], x) #берем минимум, поскольку
            #нас интересует минимально возможная сумма
        s = [x for x in s1 if x != 0]
    print(min(x for x in s if x%9 != 0))
```

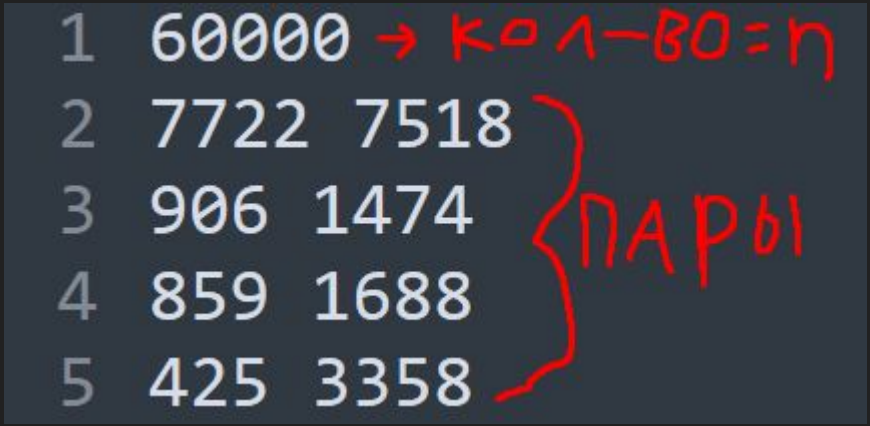
Тройки и пары #10: упрощаем МЧС

Пришло время МЧС! В прошлых блоках разобрались, как он работает, писали длинный и полный код. В этом блоке рассмотрим, как можно упростить МЧС (написание кода для его реализации), таким образом экономить свое время, силы и нервы...

Рассмотрим задачу...

(№ 2660) (Демовариант 2021 г.). Имеется набор данных, состоящий из пар положительных целых чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма всех выбранных чисел не делилась на 3 и при этом была максимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимально возможную сумму, соответствующую условиям задачи.

Что в файлике? ⇔



Пишем короткий код для МЧС...

Создадим буфер накопленных сумм ⇔ s = [0]

for i in range(n): ⇔ проходимся по циклу, далее…

pair = list(map(int, f.readline().split())) ⇔ прочитали пару

combinations = [a + b for a in s for b in pair] ⇔ образовали всевозможные комбинации пар, доступные нам вместе с уже сохраненными суммами в s и суммами в pair; далее преобразовываем буфер s в ТЕКУЩИЙ СЛОВАРЬ, в котором сортируем максимальные (или минимальные, в зависимости от условия задачи) суммы с соответствующими остатками, которые есть в combinations:

s = {x%3: x for x in sorted(combinations)}.values()

Метод values() позволяет нам передавать в s только значения, отбросив ключ словаря, в нашем случае ключ равен остатку. Почему sorted(combinations)?

⇔ ищем наибольшую сумму, следовательно, sorted(); а если наименьшую?

⇔ sorted(reverse = True) ⇔ сортировка с переворотом (reverse)

Получаем результат

В итоге в буфере s хранятся наибольшие суммы, которые удалось образовать, с соответствующими остатками. Выбираем сумму с тем остатком, который нас интересует, после чего выводим результат:

```
print(max(x for x in s if x%3 != 0))
```

Как мы можем заметить, решение стало намного проще, а код куда короче. На следующем слайде сможете увидеть, насколько проще стал код. Забавно!

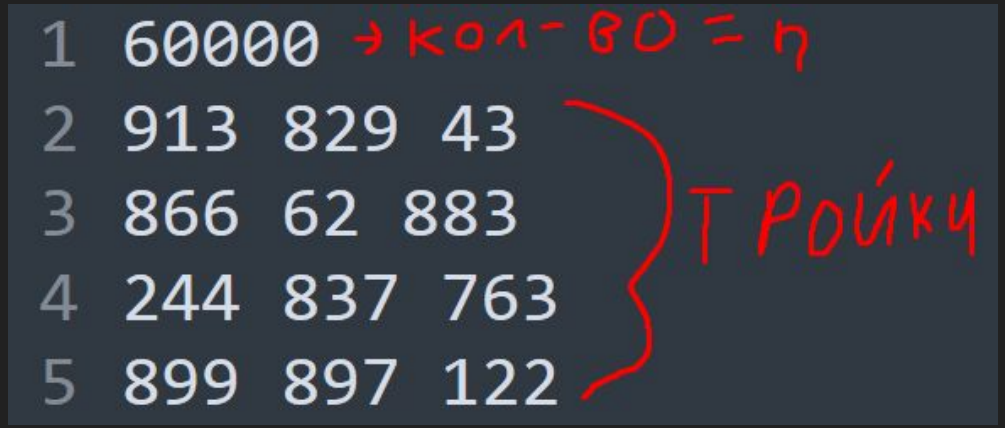
Рассмотрим полное решение на Python 3:

```
with open('27-b (1).txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        pair = list(map(int, f.readline().split()))
        combinations = [a + b for a in s for b in pair]
        s = {x%3: x for x in sorted(combinations)}.values()
    print(max(x for x in s if x%3 != 0))
```

Рассмотрим задачу посложнее...

(№ 2689) Имеется набор данных, состоящий из троек положительных целых чисел. Необходимо выбрать из каждой тройки **два числа** так, чтобы сумма всех выбранных чисел не делилась на 5 и при этом была максимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимально возможную сумму, соответствующую условиям задачи.

Что в файлике? ⇔



Пишем короткий код для МЧС...

Создадим буфер накопленных сумм ⇔ s = [0]

for i in range(n): ⇔ проходимся по циклу, далее…

a, b, c = list(map(int, f.readline().split())) ⇔ прочитали тройку

pairs = [a + b, a + c, b + c] ⇔ сгенерировали всевозможные суммы пар чисел из тройки

combinations = [a + b for a in s for b in pairs] ⇔ образовали всевозможные комбинации сумм, доступные нам вместе с уже сохраненными суммами в s и суммами в pairs; далее преобразовываем буфер s в ТЕКУЩИЙ СЛОВАРЬ, в котором сортируем максимальные (или минимальные, в зависимости от условия задачи) суммы с соответствующими остатками, которые есть в combinations:

s = {x%5: x for x in sorted(combinations)}.values()

Метод values() позволяет нам передавать в s только значения, отбросив ключ словаря, в нашем случае ключ равен остатку. Почему sorted(combinations)?

⇔ ищем наибольшую сумму, следовательно, sorted()

Получаем результат

В итоге в буфере s хранятся наибольшие суммы, которые удалось образовать, с соответствующими остатками. Выбираем сумму с тем остатком, который нас интересует, после чего выводим результат:

```
print(max(x for x in s if x%5 != 0))
```

Рассмотрим полное решение на Python 3:

```
with open('27-29b.txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        a, b, c = list(map(int, f.readline().split()))
        pairs = [a + b, a + c, b + c]
        combinations = [a + b for a in s for b in pairs]
        s = {x%5: x for x in sorted(combinations)}.values()
    print(max(x for x in s if x%5 != 0))
```

Тройки и пары #10: практика - задачи

Решим легкие и средние задачи, связанные с упрощенным МЧС (методом “частичных сумм”). Закрепим изученный материал на практике...

#10: №1

(№ 3201) (А. Куканова) Дана последовательность, которая состоит из пар натуральных чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма всех выбранных чисел имела такой же остаток от деления на 7, как наименьшая возможная, и при этом была максимальной возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимальную возможную сумму, соответствующую условиям задачи.

Здесь, помимо основной суммы, найдем еще и минимально возможную, дабы сравнить остатки: mins ⇔ минимальная сумма, тогда будем выводить результат:

```
print(max(x for x in s if x%7 == mins%7))
```

Решение задачи на Python 3:

```
with open('27-46b.txt') as f:
    n = int(f.readline())
    s = [0]
    mins = 0 #минимально возможная сумма
    for i in range(n):
        pair = list(map(int, f.readline().split()))
        mins += min(pair) #наращиваем минимальную сумму
        combinations = [a + b for a in s for b in pair]
        s = {x%7: x for x in sorted(combinations)}.values()
    #выводим сумму с таким же остатком от дел. на 7, как и у mins
    print(max(x for x in s if x%7 == mins%7))
```

#10: №2

(№ 2694) Имеется набор данных, состоящий из троек положительных целых чисел. Необходимо выбрать из каждой тройки **два числа** так, чтобы сумма всех выбранных чисел делилась на 6 и при этом была минимально возможной. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – минимально возможную сумму, соответствующую условиям задачи.

Обратим внимание на то, что нас интересует минимально возможная сумма, значит, сортируем с переворотом ⇔ sorted(reverse = True). Далее выводим результат:

```
print(min(x for x in s if x%6 == 0))
```

Решение задачи на Python 3:

```
with open('27-34b.txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        a, b, c = list(map(int, f.readline().split()))
        pairs = [a + b, a + c, b + c]
        combinations = [a + b for a in s for b in pairs]
        #тут ОБЯЗАТЕЛЬНО РЕВЕРСИМ, мы ищем МИНИМУМ!
        s = {x%6: x for x in sorted(combinations, reverse = True)}.values()
    print(min(x for x in s if x%6 == 0))
```

#10: №3

(№ 4201) (В. Ярцев) Имеется набор данных, состоящий из троек положительных целых чисел. Необходимо выбрать из каждой тройки ровно два числа так, чтобы сумма всех выбранных чисел делилась на 3 или на 17, но не делилась на оба этих числа одновременно, и при этом была минимально возможной. Гарантируется, что искомую сумму получить можно.

условиям задачи.

Вспоминаем блоки #2 и #4, друзья! Чтобы сумма делилась на x и на y одновременно, нужно, чтобы она делилась на x*y. В нашем случае нужно, чтобы не делилась, значит, храним остатки 3*17 = 51. Далее выводим результат, все по классике:

```
print(min(x for x in s if (x%3 == 0 or x%17 == 0) and x%51 != 0))
```

Решение задачи на Python 3:

```
with open('27-68b.txt') as f:
    n = int(f.readline())
    s = [0]
    for i in range(n):
        a, b, c = list(map(int, f.readline().split()))
        pairs = [a + b, a + c, b + c]
        combinations = [a + b for a in s for b in pairs]
        #И ЗДЕСЬ РЕВЕРСИМ! ОПЯТЬ НУЖНА МИНИМАЛЬНАЯ СУММА!
        s = {x%51: x for x in sorted(combinations, reverse = True)}.values()
    print(min(x for x in s if (x%17 == 0 or x%3 == 0) and x%51 != 0))
```

Группы чисел #11: поиск количеств / сумм

Ранее, в блоках #2 - #6 мы встречались с задачами на поиск количества пар чисел по определенным критериям. Существуют такие же задачи, только найти теперь нужно не количество пар, сумма которых кратна чему-то, а количество групп чисел, размер которых не определен, т. е., элементов в группе может быть как 2, так и 7, так и 137. В рамках этого, #11, блока, массивы - наши лучшие друзья...

Рассмотрим задачу...

(№ 3822) (А. Кабанов) В файле записана последовательность натуральных чисел. Гарантируется, что все числа различны. Рассматриваются всевозможные группы чисел, состоящие из любого количества элементов последовательности. Необходимо найти количество таких групп, для которых сумма элементов кратна 3.

Как будем решать? Создаем массив по остаткам, в котором будем хранить количество чисел и групп чисел со всевозможными остатками от деления на 3:

```
k = [0]*3
```

В итоге нас будет интересовать k[0] ⇔ количество групп с остатком % 3 = 0

Как будем наращивать количество?

for i in range(n): ⇔ проходимся по файлу

    x = int(f.readline()) ⇔ считываем очередное число

    knew = [0]*3 ⇔ текущий массив количества (на n - i шаге)

    knew[x%3] += 1 ⇔ увеличиваем количество по остатку (см. #2; #4)

    for y in range(3): ⇔ проходимся по остаткам

        knew[(x + y)%3] += k[y] ⇔ образовываем количество групп с новым числом x с соответствующими остатками

    for y in range(3): ⇔ проходимся по остаткам

        k[y] += knew[y] ⇔ сохраняем в основной буфер k вновь полученную информацию из knew[y] с соответствующими остатками

Рассмотрим полное решение на Python 3:

```
with open('27-58b.txt') as f:
    n = int(f.readline())
    k = [0]*3
    for i in range(n):
        x = int(f.readline())
        knew = [0]*3
        knew[x%3] += 1
        for y in range(3):
            knew[(x+y)%3] += k[y]
        for y in range(3):
            k[y] += knew[y]
    print(k[0])
```

Рассмотрим задачу...

(№ 3824) (А. Кабанов) В файле записана последовательность натуральных чисел. Гарантируется, что все числа различны. Рассматриваются всевозможные группы чисел, состоящие из любого количества элементов последовательности. Необходимо найти наибольшую сумму такой группы, кратную 25. Программа должна вывести эту сумму.

Как будем решать? Создаем массив с максимальными суммами по остаткам:

```
k = [0]*25
```

В итоге нас будет интересовать k[0] ⇔ максимал. сумма с остатком % 25 = 0

Как будем наращивать сумму?

```
for i in range(n): ⇔ проходимся по файлу
    x = int(f.readline()) ⇔ считываем очередное число
    knew = [0]*25 ⇔ текущий массив максимал. сумм (на n - i шаге)
    for y in range(25):
        knew[y] = k[y] ⇔ копируем данные
    for y in range(25): ⇔ проходимся по остаткам
        if k[y] != 0: ⇔ если ячейка не пустая
            knew[(x + y)%25] = max(knew[(x + y)%25], k[y] + x) ⇔ перезаписываем максимум
    if knew[x%25] == 0: ⇔ если вообще нет таких элементов (пусто)
        knew[x%25] = x ⇔ тогда пусть будет x, начальный элемент для наращивания
    for y in range(25): ⇔ проходимся по остаткам
        k[y] = knew[y] ⇔ сохраняем в основной буфер k вновь полученную информацию из knew[y] с
соответствующими остатками
```

Рассмотрим полное решение на Python 3:

```
with open('27-60b.txt') as f:
    n = int(f.readline())
    k = [0]*25
    for i in range(n):
        x = int(f.readline())
        knew = [0]*25
        for y in range(25):
            knew[y] = k[y]
        for y in range(25):
            if k[y] != 0:
                knew[(x + y)%25] = max(knew[(x + y)%25], k[y] + x)
        if knew[x%25] == 0:
            knew[x%25] = x
        for y in range(25):
            k[y] = knew[y]
    print(k[0])
```

Группы чисел #11: практика - задачи

Решим легкие и средние задачи, связанные с обработкой групп чисел любой размерности. Закрепим изученный материал на практике...

#11: №1

(№ 3823) (А. Кабанов) В файле записана последовательность натуральных чисел. Гарантируется, что все числа различны. Рассматриваются всевозможные группы чисел, состоящие из любого количества элементов последовательности. Необходимо найти количество таких групп, для которых сумма элементов оканчивается на 5.

k = [0]*10 ⇔ всего на 10 разных вариантов может оканчиваться сумма элементов: (0...9), дальше по накатанной, все аналогично

Решение задачи на Python 3:

```
with open('27-59b.txt') as f:
    n = int(f.readline())
    k = [0]*10
    for i in range(n):
        x = int(f.readline())
        knew = [0]*10
        for y in range(10):
            knew[y] = k[y]
        for y in range(10):
            knew[(x + y)%10] += k[y]
        knew[x%10] += 1
        for y in range(10):
            k[y] = knew[y]
    print(k[5]) #выводим пятую ячейку, остаток соответствует индексу
```

#11: №2

(№ 3825) (А. Кабанов) В файле записана последовательность натуральных чисел. Гарантируется, что все числа различны. Рассматриваются всевозможные группы чисел, состоящие из любого количества элементов последовательности. Необходимо найти наибольшую сумму такой группы, заканчивающуюся на 50. Программа должна вывести эту сумму.

k = [0]*100 ⇔ всего на 100 разных вариантов может оканчиваться сумма элементов, если мы рассматриваем два последних числа (в нашем случае это так): (0...99), дальше по накатанной, все аналогично

Решение задачи на Python 3:

```
with open('27-61b.txt') as f:
    n = int(f.readline())
    k = [0]*100
    for i in range(n):
        x = int(f.readline())
        knew = [0]*100
        for y in range(100):
            knew[y] = k[y]
        for y in range(100):
            if k[y] != 0:
                knew[(x + y)%100] = max(knew[(x + y)%100], k[y] + x)
        if knew[x%100] == 0:
            knew[x%100] = x
        for y in range(100):
            k[y] = knew[y]
    print(k[50]) #выводим пятидесятую ячейку, остаток соответствует индексу
```

Распределение по группам #12: остатки

В этом блоке познакомимся с задачами, в которых нам даны два или три ряда чисел (пары или тройки чисел), и нужно распределить их на группы так, чтобы соблюдались некоторые условия (чётность / нечётность, сравнение, соответствие и т.д.). Задачи неприятные, замечены были только на пробниках от Статграда, можно полагать, что на ЕГЭ похожие точно не встретятся, но, тем не менее, следует разобрать...

Рассмотрим задачу...

(№ 3146) Дана последовательность, которая состоит из троек натуральных чисел. Необходимо распределить все числа на три группы, при этом в каждую группу должно попасть ровно одно число из каждой исходной тройки. Сумма всех чисел в первой группе должна быть нечётной, во второй – чётной. Определите максимально возможную сумму всех чисел в третьей группе.

Как будем решать? Распределим все числа на три группы, при этом будем сохранять минимальные разности по остаткам, которые в конечном счете будем вычитать из итоговой суммы, чтобы добиться соблюдаемости условий.

k = [] ⇔ список сохраненных разностей

Находим суммы в группах

next(f) ⇔ пропускаем первое число в файле, оно нам не понадобится

s1 = s2 = s3 = 0 ⇔ храним здесь все три суммы

k = [] ⇔ список минимальных разностей

for el in f: ⇔ проходимся по файлику

    a = sorted(list(map(int, el.split()))) ⇔ прочитали три числа и отсортировали по возрастанию

    s1 += a[0] ⇔ наращиваем сумму чисел в первой группе

    s2 += a[1] ⇔ наращиваем сумму чисел во второй группе

    s3 += a[2] ⇔ наращиваем сумму чисел в третьей группе

Далее нужно как-то заполнять k, как логичнее всего это сделать?

Сохраняем минимальные разности

if a[1]%2 != a[0]%2: ⇔ если элементы с разными остатками

k.append([(a[1] - a[0]), a[0], a[1]]) ⇔ добавляем разность и каждый из элементов 1 и 2

if a[2]%2 != a[0]%2: ⇔ если элементы с разными остатками

k.append([(a[2] - a[0]), a[0], a[2]]) ⇔ добавляем разность и каждый из элементов 1 и 3

if a[2]%2 != a[1]%2: ⇔ если элементы с разными остатками

k.append([(a[2] - a[1]), a[1], a[2]]) ⇔ добавляем разность и каждый из элементов 2 и 3

k.sort() ⇔ сортируем по возрастанию, ведь нас интересуют МИНИМАЛЬНЫЕ разности

Вычитаем до соответствия

n = 0 ⇔ счетчик элементов в списке (индексатор)

while True: ⇔ пока не выполнен брейк (не достигнута позиция окончания)

if (s1 - k[n][1]%2 != 0) and (s2 - k[n][2])%2 == 0): ⇔ смотрим, если из суммы чисел в первой группе вычесть соответствующий элемент минимальной разности, и она будет нечетна, и если из суммы чисел во второй группе вычесть соответственно второй элемент, и она будет четна (это соответствует условию, сумма чисел в первой группе должна быть четна, а во второй нечетна)

print(s3 - k[n][0]) ⇔ тогда вычитаем эту минимальную разность из суммы чисел в третьей группы и выводим ответ

break ⇔ выходим из бесконечного цикла

n += 1 ⇔ наращиваем индексатор

Рассмотрим полное решение на Python 3:

```
with open('27-40b.txt') as f:
    next(f)

    s1 = s2 = s3 = 0
    k = []

    for el in f:
        a = sorted(list(map(int, el.split())))
        s1 += a[0]
        s2 += a[1]
        s3 += a[2]
        if a[1]%2 != a[0]%2:
            k.append([(a[1] - a[0]), a[0], a[1]])
        if a[2]%2 != a[0]%2:
            k.append([(a[2] - a[0]), a[0], a[2]])
        if a[2]%2 != a[1]%2:
            k.append([(a[2] - a[1]), a[1], a[2]])
    k.sort()

    n = 0
    while True:
        if (s1 - k[n][1]%2 != 0 and (s2 - k[n][2])%2 == 0):
            print(s3 - k[n][0])
            break
        n += 1
```

Рассмотрим еще одну задачу...

(№ 3701) Набор данных состоит из нечётного количества пар натуральных чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма выбранных чисел была максимальной при условии, что чётность этой суммы **НЕ совпадает** с чётностью большинства выбранных чисел. Определите максимальную сумму, которую можно получить при таком условии. Гарантируется, что удовлетворяющий условиям выбор возможен.

Как будем решать? Будем сохранять всякие разности по остаткам, которые в конечном счете будем вычитать из итоговой суммы, чтобы добиться соблюдаемости условий, при этом будем сохранять количество учтенных четных и нечетных элементов.

k = [0]*2 ⇔ количество чет. / нечет.

mini = [] ⇔ список сохраненных разностей

Накапливаем сумму

k = [0]*2 ⇔ количество чет. / нечет.

mini = [] ⇔ список разностей

s = 0 ⇔ максимальная сумма (накопленная)

for i in range(n): ⇔ проходимся по файлу

    x, y = map(int, f.readline().split()) ⇔ прочитали пару чисел

    s += max(x, y) ⇔ наращиваем сумму на максимальное число

Теперь нужно как-то обрабатывать четность и нечетность выбранных элементов, добавлять при этом разности в mini

Учитываем четность, находим разности

На каждой итерации по циклу подсчитываем количество четных и нечетных выбранных максимальных чисел:

k[(max(x, y))%2] += 1 ⇔ считаем количество чисел по остаткам

if x%2 != y%2: ⇔ если четность чисел в паре не совпадает

    mini.append(abs(x - y)) ⇔ сохраняем их разность по модулю

Теперь остается проверить четность / нечетность и вывести результат:

if max(k) != k[s%2]: ⇔ если четность большинства выбранных чисел не равна четности суммы

    print(s) ⇔ тогда выводим результат

else: ⇔ иначе

    print(s - min(mini)) ⇔ вычитаем минимальную разность, в результате четность суммы меняется, получаем верный ответ

Рассмотрим полное решение на Python 3:

```
with open('27-50a.txt') as f:
    n = int(f.readline())

    k = [0]*2
    s = 0
    mini = []

    for i in range(n):
        x, y = map(int, f.readline().split())
        s += max(x, y)
        k[(max(x, y))%2] += 1
        if (x%2 != y%2):
            mini.append(abs(x - y))

    if max(k) != k[s%2]:
        print(s)
    else:
         print(s - min(mini))
```

Распределение по группам #12: практика - задачи

Решим легкие и средние задачи, связанные с обработкой групп по условиям четности и нечетности количества закрепленных за ними элементов. Закрепим изученный материал на практике...

#12: №1

(№ 3147) Дана последовательность, которая состоит из троек натуральных чисел. Необходимо распределить все числа на три группы, при этом в каждую группу должно попасть ровно одно число из каждой исходной тройки. Сумма всех чисел в первой группе должна быть нечётной, во второй – чётной. Определите минимально возможную сумму всех чисел в третьей группе.

Ровно та же логика: ищем по остаткам непохожие минимальные разности, после чего по циклу вычитаем эти разности с сортировкой от минимального к максимальному, как только условие соблюдается, делаем брэйк ⇔ получаем в результате наибольшую сумму

k = [] ⇔ список сохраненных разностей

Решение задачи на Python 3:

```
with open('27-41b.txt') as f:
    next(f)

    s1 = s2 = s3 = 0
    k = []

    for el in f:
        a = sorted(list(map(int, el.split())))
        s3 += a[0]
        s2 += a[1]
        s1 += a[2]
        if a[1]%2 != a[0]%2:
            k.append([(a[1] - a[0]), a[0], a[1]])
        if a[2]%2 != a[0]%2:
            k.append([(a[2] - a[0]), a[0], a[2]])
        if a[2]%2 != a[1]%2:
            k.append([(a[2] - a[1]), a[1], a[2]])
    k.sort()

    n = 0
    while True:
         if (s1 - k[n][1])%2 == 0 and (s2 - k[n][2])%2 != 0:
            print(s3 + k[n][0])
            break
         n += 1
```

#12: №2

(№ 3702) Набор данных состоит из нечётного количества пар натуральных чисел. Необходимо выбрать из каждой пары ровно одно число так, чтобы сумма выбранных чисел была минимальной при условии, что чётность этой суммы **НЕ совпадает** с чётностью большинства выбранных чисел. Определите минимальную сумму, которую можно получить при таком условии. Гарантируется, что удовлетворяющий условиям выбор возможен.

Ровно та же логика: высчитываем четность большинства выбранных элементов, если четность итоговой суммы ей равна, тогда прибавляем минимальную разность с различными остатками, в результате чего четность суммы меняется ⇔ имеем наибольшую подходящую сумму

k = [0]*2 ⇔ счетчик четности

mini = [] ⇔ список разностей

Решение задачи на Python 3:

```
with open('27-51a.txt') as f:
    n = int(f.readline())

    k = [0]*2
    s = 0
    mini = []

    for i in range(n):
        x, y = map(int, f.readline().split())
        s += min(x, y)
        k[(max(x, y))%2] += 1
        if (x%2 != y%2):
            mini.append(abs(x - y))

    if max(k) != k[s%2]:
        print(s)
    else:
         print(s + min(mini))
```

Подпоследовательности #13: длины / суммы

В этом блоке познакомимся с задачами, в которых нам дана последовательность целых чисел, которую предстоит обработать, в результате чего найти самую длинную / самую короткую непрерывную подпоследовательность, в которой соблюдены некоторые условия. Иногда нужно найти сумму элементов в ней, иногда найти ее длину, существуют разные задачи, будем разбираться. Кстати, задача подобного типа встретилась на основной волне КЕГЭ-2021...

Рассмотрим задачу...

(№ 4281) (демо-2022) Дана последовательность из N натуральных чисел. Рассматриваются все её непре-рывные подпоследовательности, такие что сумма элементов каждой из них кратна k = 43. Найдите среди них подпоследовательность с максимальной суммой, определите её длину. Если таких подпоследовательностей найдено несколько, в ответе укажите количество элементов самой короткой из них.

Как будем решать? МАССИВЫ - МОЩЬ! Заведем соответствующие по индексам массивы, отвечающие за разные функции: один будет отвечать за сумму какого-то количества чисел, второй за количество чисел, которые, собственно, эту сумму образовали

prefs = [0]*43 ⇔ массив сумм с разными остатками от 43

prefl = [0]*43 ⇔ массив длин выше сохраненных сумм

Накапливаем сумму

maxs = 0 ⇔ максимальная сумма, кратная 43

minl = float(‘inf’) ⇔ минимальная длина максимальной суммы

s = 0 ⇔ накапливаемая сумма

for i in range(n): ⇔ проходимся по файлику

    x = int(f.readline()) ⇔ читаем число

    s += x ⇔ наращиваем текущую сумму

    ost = s%43 ⇔ остаток наращенной суммы на шаге (n - i)

Теперь нужно как-то обрабатывать массивы...

Обрабатываем массивы

Идея: мы будем находить минимальные суммы идущих подряд чисел с разными остатками от деления на 43, далее мы из каждой последующей суммы на каждом шаге будем отнимать минимальную сумму с таким же остатком (и по теоории чисел, что изучали в блоках #2 - #4, мы знаем, что будем получать сумму с остатком = 0, т.е., кратную, подходящую нам), так мы в результате будем находить максимальную сумму, кратную 43. Для этого нам нужно найти минимальные суммы и количество элементов, в них содержащихся (их длины):

if prefs[ost] == 0: ⇔ если сумма с таким остатком не определена

prefs[ost] = s ⇔ определяем ее равной s (текущей наращенной сумме, она в любом случае минимальна, т.к. до этого сумм не встретилось)

prefl[ost] = i + 1 ⇔ записываем длину этой самой s в массив длин

Как теперь обновлять максимальную сумму и длину?

Обновляем макс. сумму и ее длину

На каждой итерации цикла:

if prefs[ost] != 0: ⇔ если минимальная сумма существует

if (s - prefs[ost]) > maxs or ((s - prefs[ost]) == maxs and (i - prefl[ost] + 1) < minl): ⇔ если текущая сумма минус сумма с таким же остатком (такая сумма, кратная 43), больше максимальной суммы, сохраненной до этого момента, или если она ей равна, но длина этой последовательности меньше (последовательность определяется как (i - prefl[ost]), ибо мы вычитаем минимальную сумму, значит, и количество элементов в итоговой сумме уменьшается на то, которое содержала в себе минимальная сумма. Если меньше, тогда:

maxs = s - prefs[ost] ⇔ обновляем максимальную сумму

minl = i - prefl[ost] + 1 ⇔ обновляем ее длину

Дорешиваем до конца

Не забываем обрабатывать исключительный случай, когда сумма на данном этапе уже кратна 43:

if ost == 0: ⇔ если сумма сразу кратна 43

    is s > maxs: ⇔ если она больше максимальной

        maxs = s ⇔ перезаписываем сумму

        minl = i + 1 ⇔ перезаписываем длину суммы

Теперь можем выводить результат, задача решена:

print(minl)

Рассмотрим полное решение на Python 3:

```
with open('27-75b.txt') as f:
    n = int(f.readline())
    prefs = [0]*43
    prefl = [0]*43
    maxs = s = 0
    minl = 0
    for i in range(n):
        x = int(f.readline())
        s += x
        ost = s%43
        if ost == 0:
            if s > maxs:
                maxs = s
                minl = i + 1
        if prefs[ost] != 0:
            if  (s - prefs[ost]) > maxs or ((s - prefs[ost]) ==  maxs and (i - prefl[ost] + 1 < minl)):
                maxs = s - prefs[ost]
                minl = i - prefl[ost] + 1
        if prefs[ost] == 0:
            prefs[ost] = s
            prefl[ost] = i + 1
    print(minl)
```

Рассмотрим еще одну задачу...

(№ 4278) (А. Кабанов) Набор данных представляет собой последовательность натуральных чисел. Необходимо найти количество подпоследовательностей подряд идущих чисел, сумма которых делится на 71. Гарантируется, что такие подпоследовательности существуют.

Как будем решать? МАССИВЫ - МОЩЬ! Заведем массив, который будет отвечать за количество сумм последовательностей с соответствующим остатком от деления на 71

k = [0]*71 ⇔ массив количеств сумм с разными остатками от 71

count, s = 0, 0 ⇔ счетчик подпоследовательностей, сумма

Обрабатываем массив

for i in range(n): ⇔ проходимся по файлику

x = int(f.readline()) ⇔ читаем число

s += x ⇔ наращиваем текущую сумму

count += k[s%71] ⇔ увеличиваем счетчик на текущее количество сумм с таким же остатком, ведь при вычитании они дадут 0 и итоговая сумма будет кратна 71, вспоминаем логику и с прошлой задачи и блоки #2 - #4 по остаткам

k[s%71] += 1 ⇔ увеличиваем количество сумм по такому остатку (по остатку такой суммы на текущем шаге)

Остается лишь обработать исключительный случай, после чего мы получим ответ

Дорешиваем до конца

if s%71 == 0: ⇔ если текущая сумма кратна 71

    count += 1 ⇔ увеличиваем счетчик

print(count) ⇔ выводим результат

Сложно? А сходства с задачами из блоков #2 - #4 не заметили? Кажется, лишь немного сложнее, но опять же нас выручают друзья-массивы!

Рассмотрим полное решение на Python 3:

```
with open('27-72a.txt') as f:
    n = int(f.readline())

    k = [0]*71
    count = s = 0

    for _ in range(n):
        x = int(f.readline())
        s += x
        if s%71 == 0:
            count += 1

        count += k[s%71]
        k[s%71] += 1

    print(count)
```

Подпоследовательности #13: практика - задачи

Решим легкие и средние задачи, связанные с обработкой непрерывных подпоследовательностей по различным критериям, с последующим сохранением их длины / суммы и т. д.

#13: №1

(№ 4279) (А. Кабанов) Набор данных представляет собой последовательность натуральных чисел. Необходимо выбрать такую подпоследовательность подряд идущих чисел, чтобы их сумма была макси-мальной, делилась на 93 и была нечётной. Гарантируется, что такая подпоследовательность существует. В качестве ответа укажите сумму чисел данной подпоследовательности.

Ровно та же логика, что и в прошлых задачах - друзья массивы! Создаем массив минимальных сумм с различными остатками от деления на 93:

k = [0]*93 ⇔ минимальные суммы с ост. от 93

mxs, s = 0, 0 ⇔ макс. сумма, текущая сумма

Решение задачи на Python 3:

```
with open('27-73b.txt') as f:
    n = int(f.readline())

    k = [float('inf')]*93
    mxs = s = 0

    for _ in range(n):
        x = int(f.readline())
        s += x
        if s%93 == 0:
            mxs = max(mxs, s)
        else:
            if k[s%93] != float('inf'):
                if (s - k[s%93])%2 != 0:
                    mxs = max(mxs, s - k[s%93])
        k[s%93] = min(k[s%93], s)

    print(mxs)
```

#13: №2

(№ 4212) Набор данных представляет собой последовательность натуральных чисел. Необходимо выбрать такую подпоследовательность подряд идущих чисел, чтобы их сумма была максимальной и делилась на 69, и определить её длину. Гарантируется, что такая подпоследовательность существует. Если таких подпоследовательностей несколько, нужно выбрать подпоследовательность наименьшей длины.

Ровно та же логика, что и в прошлых задачах - друзья массивы! Создаем соответствующие массивы с суммами и длинами

prefs = [0]*69 ⇔ минимальные суммы с ост. от 69

prefl = [0]*69 ⇔ длины этих самых сумм

Решение задачи на Python 3:

```
with open('27-71b.txt') as f:
    n = int(f.readline())
    prefs = [0]*69
    prefl = [0]*69
    maxs = s = 0
    minl = 0
    for i in range(n):
        x = int(f.readline())
        s += x
        ost = s%69
        if ost == 0:
            if s > maxs:
                maxs = s
                minl = i + 1
        if prefs[ost] != 0:
            if  (s - prefs[ost]) > maxs or ((s - prefs[ost]) ==  maxs and (i - prefl[ost] + 1 < minl)):
                maxs = s - prefs[ost]
                minl = i - prefl[ost] + 1
        if prefs[ost] == 0:
            prefs[ost] = s
            prefl[ost] = i + 1
    print(minl)
```

Переборные алгоритмы #14: вложенные циклы

В этом блоке разберемся, как решать задачи переборно (неэффективно), переборным алгоритмом с использованием вложенных (нескольких зависимых друг от друга) циклов. Такое решение не подойдет для файла B, однако, может выручить в некоторых ситуациях, когда необходимо найти ответ для файла A, но оптимальное решение придумать не удалось...

Рассмотрим на примере:

```
a = [3, 5, 7, 11] ⇔ список из 4 чисел
n = len(a) ⇔ длина списка
for i in range(n - 1): ⇔ 1-й цикл
    for j in range(i + 1, n): ⇔ 2-й цикл
        print(a[i] + a[j]) ⇔ вывод суммы
```

Какие значения будут принимать i, j?

Hi, j - итераторы цикла. Вы точно знаете, что i примет значения от 0 до (n - 1), где n - количество чисел в списке. Т.е., в нашем случае, i = 0; i = 1;i = 2; какие тогда значения примет j? Логично, от (i + 1), но до n (это мы прописали в коде, см. пред. слайд), т.е. j = 1; j = 2; j = 3, НО сколько раз j примет эти значения? ⇔ (n - 1) раз ⇔ столько же раз, сколько раз будет изменять значение i (сколько различных значений у i), т.е., j будет принимать (n - 1) значений какое-то количество раз, в нашем случае (n - 1) раз, это прямо зависит от i, ибо цикл с j является вложенным для цикла с i. Т.е., j примет различные значения ((n - 1) * (n - 1)) раз. Давайте посмотрим, что будет происходить и какие значения будем иметь на выводе...

Что будет выведено?

i, j = 0, 1 ⇔ a[i] = 3, a[j] = 5 ⇔ выведено 5 + 3 = 8

i, j = 0, 2 ⇔ a[i] = 3, a[j] = 7 ⇔ выведено 7 + 3 = 10

i, j = 0, 3 ⇔ a[i] = 3, a[j] = 7 ⇔ выведено 11 + 3 = 14

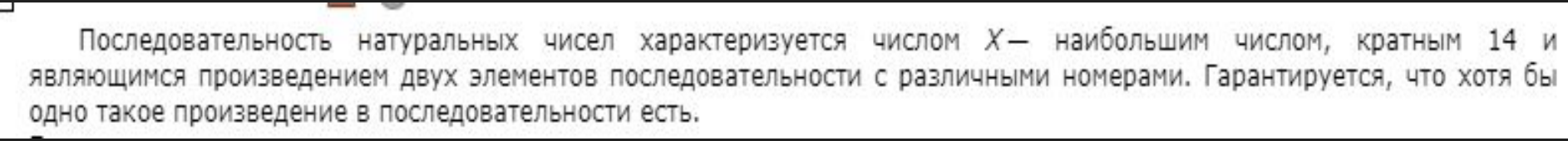
i, j = 1, 2 ⇔ a[i] = 5, a[j] = 7 ⇔ выведено 5 + 7 = 12

i, j = 1, 3 ⇔ a[i] = 5; a[j] = 11 ⇔ выведено 5 + 11 = 16

i, j = 2, 3 ⇔ a[i] = 7, a[j] = 11 ⇔ выведено 7 + 11 = 18

Как итог, мы получили суммы абсолютно всех пар, которые можно образовать из этих 4 чисел, получили мы их с помощью вложенных циклов, где цикл с j напрямую зависит от цикла с i, выполняется от i + 1 до n, при этом элементы по индексу никогда не соприкасаются (элемент не может образовать пару с самим собой), поэтому, мы и ведем отчет для j от i + 1 и до n, в свою очередь, для i ведем отчет до n - 1, чтобы элементы в точке n не совпали

Рассмотрим задачу...



Последовательность натуральных чисел характеризуется числом X — наибольшим числом, кратным 14 и являющимся произведением двух элементов последовательности с различными номерами. Гарантируется, что хотя бы одно такое произведение в последовательности есть.

Как будем решать? Сейчас решим задачу переборно, с использованием вложенных циклов, решение сработает только для файла A, ведь переборный алгоритм неэффективен. Надеюсь, понятно почему: количество итераций теперь равно не n, оно равно n * k, где k постоянно уменьшается, а в некоторых случаях оно вообще квадратично и равно n*n.

a = [int(x) for x in f] ⇔ запишем все числа из файла в список, далее будем их обрабатывать

maxi = 0 ⇔ максимальное произведение

Перебираем все произведения

for i in range(n - 1): ⇔ для каждого элемента с индексом i

    for j in range(i + 1, n): ⇔ для каждого элемента с индексом от i + 1

        if (a[i]*a[j])%14 == 0: ⇔ если произведение элементов кратно 14

            maxi = max(maxi, a[i]*a[j]) ⇔ если оно меньше, то обновляем

В итоге в maxi мы имеем наибольшее произведение двух чисел, кратное 14. Получили мы его путем перебора всевозможных произведений, которые возможно получить из пар чисел в файле. Для большого количества чисел, например, для миллиона, такой алгоритм не будет эффективен, поскольку он будет вычислять (n * ((n - 1) + (n - 2) + (n - 3) + (n - k))комбинаций, уже на первом шаге это будет = 999999 итераций, а шагов будет n. На следующем слайде посмотрим на полное решение...

Рассмотрим полное решение на Python 3:

```
with open('27-A_2.txt') as f:
    n = int(f.readline())
    a = [int(x) for x in f]
    maxi = 0
    for i in range(n - 1):
        for j in range(i + 1, n):
            if (a[i]*a[j])%14 == 0:
                maxi = max(a[i]*a[j], maxi)
    print(maxi)
```

Переборные алгоритмы #14: практика - задачи

Решим легкие и средние задачи, связанные с обработкой непрерывных подпоследовательностей по различным критериям, с последующим сохранением их длины / суммы и т. д.

#14: №1

По каналу связи передавалась последовательность положительных целых чисел, все числа не превышают 1000. Количество чисел известно. Затем передаётся контрольное значение последовательности — наибольшее число R, удовлетворяющее следующим условиям:

1) R — произведение двух различных переданных элементов последовательности («различные» означает, что не рассматриваются квадраты переданных чисел, произведения различных элементов последовательности, равных по величине, допускаются);

2) R делится на 14.

Если такого числа R нет, то контрольное значение полагается равным 0. В результате помех при передаче как сами числа, так и контрольное значение могут быть искажены.

Пробуйте свои силы, на следующем слайде будет представлено решение.

Решение задачи на Python 3 (неэффективное):

```
with open('27985_A.txt') as f:
    n = int(f.readline())
    a = [int(x) for x in f]
    maxi = 0
    for i in range(n - 1):
        for j in range(i + 1, n):
            if (a[i]*a[j])%14 == 0:
                maxi = max(a[i]*a[j], maxi)
    print(maxi)
```

#14: №1

На спутнике «Восход» установлен прибор, предназначенный для измерения солнечной активности. В течение времени эксперимента (это время известно заранее) прибор каждую минуту передаёт в обсерваторию по каналу связи положительное целое число, не превышающее 1000, — количество энергии солнечного излучения, полученной за последнюю минуту, измеренное в условных единицах.

После окончания эксперимента передаётся контрольное значение — наибольшее число R, удовлетворяющее следующим условиям:

1) R — произведение двух чисел, переданных в разные минуты;

2) R делится на 26.

Необходимо найти такое число R. Предполагается, что удовлетворяющее условиям контрольное значение существовало в момент передачи. В результате помех при передаче как сами числа, так и контрольное значение могут быть искажены.

Если удовлетворяющее условию контрольное значение определить невозможно, то выводится число 0.

На вход программе в первой строке подаётся количество чисел $1 < N \le 100\,000$. В каждой из последующих N строк записано одно положительное целое число, не превышающее 1000.

Здесь Вы, наверное, ждете какой-то подсказки, но я не знаю, что можно придумать)))) Все слишком легко и очевидно, пробуйте написать неэффективное решение самостоятельно, далее будет решение.

Решение задачи на Python 3 (неэффективное):

```
with open('27988_A.txt') as f:
    n = int(f.readline())
    a = [int(x) for x in f]
    maxi = 0
    for i in range(n - 1):
        for j in range(i + 1, n):
            if (a[i]*a[j])%26 == 0:
                maxi = max(a[i]*a[j], maxi)
    print(maxi)
```

Бонусный блок #15: экзотика (задачи и решения)

В этом блоке разберем некоторые экзотические задачи. Важно понимать, экзотические - не значит, что сложные. Речь пойдет о задачах с интересными и нестандартными формулировками, идеями. Разумеется, среди них будут и очень сложные, так называемые дикие задачи-гробы. Такие вряд ли встретятся на ЕГЭ, ведь их не было ни на апробациях, ни на пробниках, но все-таки следует познакомиться и с ними, дабы иметь уверенность в себе. Кстати, рассмотрим не только экзотические задачи, но еще и экзотические способы решения обычных задач. Рассмотрим, например, применение “1000-IQ МЧС” (прокачка метода частичных сумм). Начинаем...

#15: №1

Начнем мы с наиболее простых задач. В некотором смысле их даже нельзя отнести к экзотическим, но и не отнесешь их к определенному типу. Эта задача на остатки, идея решения может быть такой же, как и идея решения задач на поиск сумм / количеств групп по условию (блок #11). Мы решим эту задачу проще (экзотически, хех):

(№ 3775) В файле записана последовательность натуральных чисел. Гарантируется, что все числа различны. Из этой последовательности нужно выбрать четыре числа, чтобы их сумма делилась на 6 и была наибольшей. Какую наибольшую сумму можно при этом получить?

Itertools, combinations...

Вы, очевидно, знаете, что такое combinations (permutations и product из той же серии). Чаще всего применяются itertools для решения задач 8 на комбинаторику. Если вдруг не слышали (что странно), название говорит само за себя: combinations ⇔ комбинации. Что ж, мы нашли itertools-ам’ еще одно применение - задача 27. Будем перебирать комбинации из чисел, погнали!

```
from itertools import combinations ⇔ импортируем нужную нам функцию
```

Нам теперь нужно определить, какие числа мы будем комбинировать. Ну, очевидно, с различными остатками: %6 = 1; = 2; = 3; = 4; = 5; = 0: 6 различных остатков

Создаем массивы по остаткам

Говорилось уже, массивы - друзья наши:

k1 = [0]*4 ⇔ 4 числа с остатком = 1

k2 = [0]*4 ⇔ 4 числа с остатком = 2

k3 = [0]*4 ⇔ 4 числа с остатком = 3

k4 = [0]*4 ⇔ 4 числа с остатком = 4

k5 = [0]*4 ⇔ 4 числа с остатком = 5

k0 = [0]*4 ⇔ 4 числа с остатком = 0

Догадались уже, почему массивы состоят из 4 ячеек? На самом деле это не очень важно, мы можем хранить как 4 числа, так и 10, но не больше (иначе перебор в combinations работать будет очень долго. Однако, есть тут своя приколюха: мы должны хранить не менее 4 чисел, поскольку возможна ситуация, когда мы сложим три числа с остатком 1 с четвертым числом, остаток которого = 3: 1 + 1 + 1 + 3 = 6 = 0, получим кратную сумму. Значит, мы должны хранить 3 наибольших числа с остатком 1. Та же ситуация с числами, у которых остаток 3: 3 + 3 + 3 + 3 = 12 = 0. т.е. нужно хранить 4 числа с остатком 3. В общем, чтобы точно не ошибиться, можно задать размерность с запасом (например, пусть она будет = 10). Но логику Вам объяснили.

Ищем наибольшие числа

Проходясь по циклу, читая число (всё это вы уже умеете давно):

```
if x%6 == 0: k0[0] = max(k0[0], x)

if x%6 == 1: k1[0] = max(k1[0], x)

if x%6 == 2: k2[0] = max(k2[0], x)

if x%6 == 3: k3[0] = max(k3[0], x)

if x%6 == 4: k4[0] = max(k4[0], x)

if x%6 == 5: k5[0] = max(k5[0], x)

k1.sort(); k2.sort(); k0.sort(); k4.sort(); k5.sort(); k3.sort()
```

Здесь все понятно, сохраняем максимальные числа по остаткам, но зачем сортируем?Сортируем, кстати, на каждой итерации: наибольшие числа уходят в конец, наименьшие в начало. На каждой итерации мы обновляем ячейку с наименьшим числом. Если вдруг она стала самой большей ⇔ в результате сортировки встает на последнее место, а та, что была больше нее, но меньше остальных, встает, соответственно, на ячейку с индексом 0. В результате находим максимумы

Комбинируем!

Теперь мы будем рассматривать всевозможные комбинации, состоящие из четырех чисел (образовываться они будут путем перестановок в найденных нами числах, их всего 6*4 = 24). Если сумма этих чисел будет кратна 6, мы будем ее учитывать. В итоге найдем максимальную:

```
maxi = 0 ⇔ максимальная сумма, кратная 6

osts = k1 + k2 + k3 + k4 + k5 + k0 ⇔ общий список наших чисел

for s in combinations(osts, 4): ⇔ создаем комбинации

    if sum(s)%6 == 0: ⇔ если сумма комбинации кратна 6

        maxi = max(maxi, sum(s)) ⇔ перезаписываем, если она больше
```

Остается только вывести результат, слава комбинейшнсам!

Рассмотрим полное решение на Python 3:

```
from itertools import combinations
with open('27-56.txt') as f:
    k1 = [0]*4
    k2 = [0]*4
    k3 = [0]*4
    k4 = [0]*4
    k5 = [0]*4
    k0 = [0]*4
    n = int(f.readline())
    for i in range(n):
        x = int(f.readline())
        if x%6 == 0: k0[0] = max(k0[0], x)
        if x%6 == 1: k1[0] = max(k1[0], x)
        if x%6 == 2: k2[0] = max(k2[0], x)
        if x%6 == 3: k3[0] = max(k3[0], x)
        if x%6 == 4: k4[0] = max(k4[0], x)
        if x%6 == 5: k5[0] = max(k5[0], x)
        k1.sort(); k2.sort(); k0.sort(); k4.sort(); k5.sort(); k3.sort()
    osts = k4 + k3 + k2 + k1 + k5 + k0
    maxi = 0
    for s in combinations(osts, 4):
        if sum(s)%6 == 0:
            maxi = max(maxi, sum(s))
    print(maxi)
```

Решим эту же задачу классически

k = [0]*6 ⇔ массив наибольших чисел с разными остатками

k2 = [0]*6 ⇔ массив наибольших сумм пар чисел с разными остатками

k3 = [0]*6 ⇔ массив наибольших сумм троек чисел с разными остатками

Пройдемся по циклу, прочитаем число (все это вы уже знаете):

k[x%6] = max(k[x%6], x) ⇔ обновляем массив наибольших чисел по остаткам

Мы обновили массив, состоящий из конкретных чисел по остаткам, теперь нам нужно обновлять еще массивы, в которых хранятся суммы пар чисел и троек чисел, соответственно. В результате в массиве k3 будут храниться наибольшие суммы, состоящие из 3 чисел с различными остатками. К ним мы на каждой итерации будем добавлять вновь прочитанное число. Если сумма будет наибольшей - будем ее обновлять. Остается только разобраться, как будем обновлять эти массивы?

Обновляем массивы сумм

for y in range(6): ⇔ проходимся по разным остаткам

k2[(x+y)%6] = max(k2[(x+y)%6], x+k[y]) ⇔ обновляем наибольшие суммы пар чисел по остаткам (x - текущее число, k[y] - наибольшее число, сохраненное раньше с таким же остатком)

for y in range(6): ⇔ проходимся по разным остаткам

k3[(x+y)%6] = max(k3[(x+y)%6], x+k2[y]) ⇔ обновляем наибольшие суммы троек чисел, складывая текущее число с наибольшей суммой пар чисел по остаткам

for y in range(6): ⇔ проходимся по остаткам

if (x+y)%6 == 0: ⇔ если текущее число в сочетании с таким остатком будет кратно 6, тогда

maxi = max(maxi, x+k3[y]) ⇔ перезаписываем максимум, если нужно

Рассмотрим полное решение на Python 3:

```
with open('27-56b.txt') as f:
    n = int(f.readline())

    k = [0]*6
    k2 = [0]*6
    k3 = [0]*6
    maxi = 0

    for _ in range(n):
        x = int(f.readline())

        for y in range(6):
            if (x+y)%6 == 0: maxi = max(maxi, x+k3[y])

        for y in range(6):
            k3[(x+y)%6] = max(k3[(x+y)%6], x+k2[y])

        for y in range(6):
            k2[(x+y)%6] = max(k2[(x+y)%6], x+k[y])

        k[x%6] = max(k[x%6], x)

    print(maxi)
```

#15: №2

Легкая задача, можно связать с обработкой последовательности целых чисел, обработкой подпоследовательностей. Но, к экзотике она как раз хорошо относится, так как является одной из тех задач, в которых словарь хорош как никогда, и на сей раз он используется не для МЧС, глянем:

(№ 4510) (Л. Шастин) Дана последовательность натуральных чисел. Рассматриваются все её непрерывные подпоследовательности, в которых начальное число последовательности делится на 21 и является квадратом конечного числа последовательности. Найдите длину наибольшей такой подпоследовательности.

Как будем решать?

one = set() ⇔ создаем множество, в нем будем хранить все числа, кратные 21, которые нам встретились

two = {} ⇔ создаем словарь, в нем будем хранить индекс (нужен для длины), на котором встретился тот или иной подходящий нам элемент

dl = 0 ⇔ здесь хранить будем длину

Обрабатываем словарь и множество

if x%21 == 0: ⇔ если текущий x делится на 21

if x not in one: ⇔ если его нет в множестве

one.add(x) ⇔ добавляем его

two[x] = i - 1 ⇔ в словарь добавляем индекс, на котором он нашелся (ключом будет само число, а длина будет значением!)

if x**2 in one: ⇔ если вдруг квадрат текущего числа есть во множестве, значит есть число, которое встретилось раньше (с него может начинаться подходящая подпоследовательность), и это число является квадратом текущего, значит текущее является его корнем, тогда, раз так

dl = max(dl, i - two[x**2]) ⇔ перезаписываем в длину максимум, если это возможно (максимум находим как текущий индекс i, на котором встретилось текущее число, минус индекс, на котором встретилось предыдущее число-квадрат

Рассмотрим полное решение на Python 3:

```
with open('27-79b.txt') as f:
    n = int(f.readline())

    one = set()
    two = {}
    dl = 0

    for i in range(n):
        x = int(f.readline())
        if x%21 == 0:
            if x not in one:
                one.add(x)
                two[x] = i - 1
        if x**2 in one:
            dl = max(dl, i - two[x**2])

    print(dl)
```

Рассмотрим решение без словаря:

```
f = open('27-A.txt')
n = int(f.readline())
nums = [0]*(10**5) #массив встретившихся нам чисел, кратных 21
lens = [0]*(10**5) #массив с индексами, на которых эти числа встретились
lmax = -1
for j in range (n):
    x = int(f.readline())
    if x%21 == 0:
        if nums[x] == 0:
            nums[x] = x
            lens[x] = j - 1
        if x**2 < 10001:
            if nums[x**2] != 0:
                l = j - lens[x**2]
                lmax = max(l, lmax)
print(lmax)
```

#15: №3

В этой задаче тоже мало сложного, здесь сработает типичный МЧС, просто нужно как-то обрабатывать НОД. Подумайте, как это можно сделать, я лично впервые встретившись с этой задачей довольно быстро ее осилил:

(№ 2711) (Д.Ф. Муфаззалов) Имеется набор данных, состоящий из троек положительных целых чисел. Из каждой тройки выбрали два числа и нашли их наибольший общий делитель (НОД). Затем все полученные таким образом значения НОД сложили. Определите наибольшую сумму, кратную числу 10, которая может быть получено таким образом. Гарантируется, что искомую сумму получить можно. Программа должна напечатать одно число – максимально возможную сумму, соответствующую условиям задачи.

Как будем решать?

Очевидно, воспользуемся типичным МЧС для нахождения суммы чисел. Одно “но”: нужно создавать комбинации из НОД чисел, т.е. в виде аргумента мы будем передавать не сами числа, а посчитанный заранее их НОД. Как будем считать НОД? Создадим функцию:

```
from math import gcd ⇔ gcd будет считать НОД
from itertools import combinations ⇔ используем комбинейшнс для комбинаций из пар чисел

def nod(list): ⇔ функция, она вернет нам НОД всех пар чисел
s = [] ⇔ в этот список запишем образовавшиеся НОД из пар чисел
for w in combinations(list, r = 2): ⇔ для всех комбинаций из двух чисел
                    s.append(gcd(w[0], w[1])) ⇔ добавляем их НОД
return s ⇔ возвращаем список получившихся в результате НОД
```

Далее решаем так, как и более простые задачи на применение МЧС, только в комбинации передаем не отдельные числа, а уже выполненную для них написанную нами функцию на НОД

Наращиваем сумму

k = {0} ⇔ храним здесь наши суммы (всп. блоки #8-10)

for i in range(n): ⇔ проходимся по файлику

troyka = [int(x) for x in f.readline().split()] ⇔ прочитали тройку

combo = {sum([x, y]) for x in nod(troyka) for y in k} ⇔ скомбинировали НОД’ы (из переданных в функцию чисел) с имеющимися ранее суммами в k

k = {x%10: x for x in sorted(combo)}.values() ⇔ сортируем по остатку

Вот и вся задача, просто передавали в качестве аргумента в комбинации не сами пары чисел, а НОД от этих самых пар. Теперь остается лишь вывести результат на экран...

Рассмотрим полное решение на Python 3:

```
from math import gcd
from itertools import combinations

def nod(list):
    s = []
    for w in combinations(list, r = 2):
        s.append(gcd(w[0], w[1]))
    return s

with open('27-36b.txt') as f:
    n = int(f.readline())

    k = {0}

    for i in range(n):
        troyka = [int(x) for x in f.readline().split()]
        combo = {sum([x, y]) for x in nod(troyka) for y in k}
        k = {x%10: x for x in sorted(combo)}.values()

    print(max(x for x in k if x%10 == 0))
```

#15: №4

Эта задача очень похожа на предыдущую, но здесь нам нужно находить НОК чисел, при этом в файлике записаны не тройки чисел, а группы из 2-10 чисел. Основной вопрос стоит в том, как нам реализовать функцию на НОК:

(№ 4429) (Е. Драчева) Набор данных состоит из групп натуральных чисел, каждая группа записана в отдельной строке. В любой группе содержится не менее двух чисел. Из каждой группы выбрали два числа и нашли их наименьшее общее краткое (НОК). Затем все полученные таким образом значения НОК сложили. Определите наибольшую сумму, кратную числу 5 или 7 (но не одновременно двум этим числам), которая может быть получена таким образом.

Импортируем библиотеки

from itertools import combinations as cmb ⇔ импортируем комбинейшнс (будем комбинировать по 2 числа из группы) как “cmb”, чтобы не путаться

from functools import reduce ⇔ reduce позволит нам применять некую функцию сразу ко всем элементами списка и множества (схожа с map, но для последовательности чисел)

from math import gcd ⇔ gcd импортируем для поиска НОД, он нам поможет в поиске НОК (они связаны)

Пишем функцию для поиска НОК

```
def nok(set):
return reduce(lambda x, y: x*y//gcd(x, y), set)
```

В функцию мы будем передавать различные комбинации, состоящие из двух чисел (комбинировать будем на ходу по циклу, далее рассмотрим как), далее находим НОК по логике алгоритма Евклида: произведение двух чисел делим на их НОД (он же gcd). Здесь reduce выполняет функцию map, оборачивая последовательность элементов (в нашем случае она = set), в лямбда-функцию. В лямбда функции, в свою очередь, хранится тот самый алгоритм, вычисляющий НОК от двух чисел, которые мы передаем

Наращиваем сумму

k = {0} ⇔ здесь храним наши суммы по остаткам

for i in range(n): ⇔ проходимся по файлику

b = list(map(int, f.readline().split()))[1:] ⇔ читаем группу чисел, пропуская первый элемент срезом (он отвечает за количество чисел в группе, нам будет только мешаться)

combinations = {nok({x, y}) for x, y in cmb (b, 2)} ⇔ находим комбинации от НОК двух выбранных чисел, выбираем их с помощью combinations, которые импортировали как “cmb”

s = {sum({x, y}) for x in k for y in combinations} ⇔ находим общие суммы с уже сохраненными и с суммами в combinations

k = {x%35: x for x in sorted(s)}.values() ⇔ сортируем по остаткам

Получаем результат

Остается только выводить результат. У кого-нибудь возник вопрос, почему мы сортируем по остаткам от деления на 35? ⇔ число делится на 35, если оно делится и на 5, и на 7, что противоречит условию задачи, здесь мы вспомнили, кстати, блоки #2 - #4, связанные с делимостью и остатками.

Выводим результат в соответствии с условием задачи:

```
print(max(x for x in k if (x%5 == 0 or x%7 == 0) and x%35 != 0))
```

Рассмотрим полное решение на Python 3:

```
from itertools import combinations as cmb
from functools import reduce
from math import gcd

def nok(set):
    return reduce(lambda x, y: x*y//gcd(x, y), set)

with open('27-77b.txt') as f:
    n = int(f.readline())

    k = {0}

    for i in range(n):
        b = list(map(int, f.readline().split()))[1:]
        combinations = {nok({x, y}) for x, y in cmb (b, 2)}
        s = {sum({x, y}) for x in k for y in combinations}
        k = {x%35: x for x in sorted(s)}.values()

    print(max(x for x in k if (x%5 == 0 or x%7 == 0) and x%35 != 0))
```

#15: №5

Задача на логику и соображалку. Крайне легкая, если разобраться. Отлично подойдет в качестве на “подумать как”, алгоритм решения будет нетипичный, но блин, насколько же простой. Изначально, когда я собирал базу решений всех задач 27 с сайта К. Ю. Полякова, наткнулся в интернете на непонятное для себя решение. После решил задачу сам, как мне кажется, куда более логичным образом. Попробуйте сами, а дальше решение:

(№ 2679) (А. Жуков) Имеется набор данных, состоящий из целых чисел. Необходимо определить максимальное произведение подпоследовательности, состоящей из одного или более идущих подряд элементов.

В чем идея решения?

В файле находятся различные числа, в том числе и отрицательные и нули. Произведение будет возрастать с каждым числом. Уменьшаться оно будет только в том случае, если умножить его на 0 (оно станет = 0, и в том случае, если умножить его на отрицательное число (было положительным, станет отрицательным), но если еще раз умножить на отрицательное, то минус на минус дадут в результате плюс. Мы будем наращивать произведение до тех пор, пока не встретился 0, на каждом шагу будем обновлять переменную, хранящую максимальное произведение. Встретился ноль - обнуляем текущее произведение, начинаем наращивать заново. На следующем слайде будет код с решением этой задачи, но пробуйте реализовать сначала сами

Рассмотрим полное решение на Python 3:

```
with open('27-19b.txt') as f:
    n = int(f.readline())
    Pcur = 0 #наращиваемое произведение
    Pmax = 0 #максимальное произведение
    for i in range(n):
        x = int(f.readline())
        if x == 0: #если встретился ноль, то обрываем последовательность
            Pcur = 1 #и обнуляем текущее произведение
        else: #иначе
            Pcur = Pcur * x  #наращиваем произведение
        Pmax = max(Pcur, Pmax) #сверяем с макс. на каждой итерации
    print(Pmax)
```

#15: №6

Ровно как и предыдущая, эта задача тоже на логику и соображалку. И вновь помогут нам наши друзья-массивы. Будем учитывать остатки с их помощью. Попробуйте решить самостоятельно, вспоминайте блоки #2 - #4, а на следующих слайдах разберем ее решение...

(№ 2710) (Е. Джобс) Дана последовательность N целых неотрицательных чисел. Необходимо определить количество пар положительных элементов этой последовательности, сумма которых четна, при этом между элементами пары есть хотя бы один ноль.

В чем идея решения?

Мы заведем два массива по остаткам от деления на 2, вспоминаете блоки #2 - #4? Еще можете вспомнить блок #11, там тоже имеется подобная логика. Почему два массива? В одном мы будем хранить всевозможные элементы с различными остатками, которые нам встретились, а в другом мы будем хранить только те, с которыми можно образовать пару (между которыми уже есть один ноль):

Kall = [0]*2 ⇔ массив всех чисел по остаткам

K0 = [0]*2 ⇔ массив подходящих чисел (между которыми уже есть один ноль)

Обрабатываем массивы

if x == 0: ⇔ если встретился ноль, значит

K0 = Kall.copy() ⇔ в массив подходящих чисел (с которыми можно образовать пару, между ними есть хотя бы один ноль) копируем данные из массива с информацией о кратности всех остальных чисел, встретившихся ранее

else: ⇔ если текущее число не ноль, тогда

count += K0[(2 - x%2)%2] ⇔ увеличиваем счетчик на количество подходящих чисел с “дополняющим” остатком (всп. блоки #2 - #4)

Kall[x%2] += 1 ⇔ количество всех чисел по остаткам обновляем

Рассмотрим полное решение на Python 3:

```
with open('27-35b.txt') as f:
    n = int(f.readline())
    Kall = [0]*2
    K0 = [0]*2
    count = 0
    for i in range(n):
        x = int(f.readline())
        if x == 0:
            K0 = Kall.copy()
        else:
            count += K0[(2 - x%2)%2]
            Kall[x%2] += 1
    print(count)
```

#15: №7

В блоках #7 - #10 мы познакомились с МЧС (методом частичных сумм). Теперь предлагаю глянуть, на что вообще способен этот чудесный метод. Мы его проапгрейдим, в общем, спойлер: продвинутый метод ⇔ настоящий убийца. Пришло время экзотических решений. Для начала решим задачу из демоверсии этим методом (для понимания общей идеи):

(№ 4281) (демо-2022) Дана последовательность из N натуральных чисел. Рассматриваются все её непре-рывные подпоследовательности, такие что сумма элементов каждой из них кратна k = 43. Найдите среди них подпоследовательность с максимальной суммой, определите её длину. Если таких подпоследовательностей найдено несколько, в ответе укажите количество элементов самой короткой из них.

В чем идея решения?

Мы будем использовать массив, который хранит в себе два аргумента: первый аргумент (первая ячейка) отвечает за накопленную в текущий момент максимальную сумму, кратную 43, а второй аргумент (вторая ячейка) отвечает за длину этой самой суммы (за количество элементов, которые эти сумму образовали):

k = [[0,0]] ⇔ массив, в нем храним сумму и ее длину

Smax = 0 ⇔ здесь храним максимальную сумму

Lmin = INF ⇔ а здесь храним длину этой суммы

Обрабатываем массив

combinations = [[a + x, b + 1] for a, b in k] + [[x, 1]]

⇔ на каждой итерации мы, читая новое число, образуем новые комбинации:(a + x) - комбинация прошлой суммы с текущим x, (b + 1) - длину прошлой суммы увеличиваем на единицу, так как добавился еще один элемент; [[x, 1]] - передаем сами аргументы: число и размерность (его длину = 1)

k = {x[0]%43: x for x in sorted(combinations)}.values()

⇔ сортируем по остаткам от деления на 43, в качестве ключа используем x[0], поскольку элемент x[1] отвечает за длину, а нам нужна сумма, она же = x[0]

Обновляем результат

На каждой итерации проходимся по k, здесь s - сумма, l - длина. Далее:

for s, l in k: ⇔ для суммы и длины в k

if s%43 == 0: ⇔ если сумма кратна 43

if (s > Smax or (s == Smax and (l < Lmin))): ⇔ если она больше максимальной суммы, либо если она равна максимальной сумме, но ее длина меньше (от нас просят минимальную длину, если это возможно), тогда

Smax, Lmin = s, l ⇔ обновляем максимальную сумму и длину

Почему решение эффективно?

Возможно, вы задались таким вопросом, ведь в каждой итерации мы проходимся по k. Но ведь в k всегда <= 43 элементов, именно поэтому, программа работает быстро. Вуаля, мы прокачали МЧС (метод частичных сумм). На парочке следующих, куда более сложных задачах, мы рассмотрим мощь этого метода. Для сравнения приведем классическое решение, тогда поймете, насколько же все-таки крут этот метод. Кстати, на следующем слайде можете глянуть на полное решение этой задачи...

Рассмотрим полное решение на Python 3:

```
with open('27-75a.txt') as f:
    n = int(f.readline())
    INF = float('inf')
    k = [[0,0]]
    Smax = 0
    Lmin = INF
    for i in range(n):
        x = int(f.readline())
        combinations = [[a + x, b + 1] for a, b in k] + [[x, 1]]
        k = {x[0]%43: x for x in sorted(combinations)}.values()
        for s, l in k:
            if s%43 == 0:
                if (s > Smax or (s == Smax and (l < Lmin))):
                    Smax, Lmin = s, l
    print(Lmin)
```

#15: №8

На первый взгляд - гроб, не правда ли? Эта задача является усложненной версией задачи с варианта от Статграда, который был опубликован в конце октября 2021 года. На самом деле, задача несложная. Ее можно решить как с помощью массивов (классически, см. блок #13), так и с помощью продвинутого МЧС. Мы рассмотрим именно второй способ, мне он нравится больше:

(№ 4515) Рассматриваются все её непрерывные подпоследовательности, в которых количество простых чисел кратно K = 9. Найдите наибольшую сумму такой подпоследовательности.

В чем идея решения?

Ровно та же идея, массив, хранящий два элемента: сумму и количество элементов (на этот раз количество простых чисел, входящих в эту сумму, это от нас спрашивают в условии). Для того, чтобы определять, является ли текущее число простым, заведем функцию:

```
p = lambda x: all(x%j != 0 for j in range(2, int(x**0.5) + 1))
```

Теперь по той же логике будем обновлять наш массив сумм и длин:

```
combinations = [[a + x, b + p(x)] for a, b in k] + [[x, p(x)]]
```

⇔ если число простое, к длине будет добавляться 1 (True), иначе будет добавляться 0 (False), тогда ничего не будет меняться

```
k = {x[1]%9: x for x in sorted(combinations)}.values() ⇔ сортируем
```

Обновляем результат

На каждой итерации проходимся по k:

for a, b in k:

⇔ для суммы, количества простых чисел в k

s = max(a, s) if b%9 == 0 else s

⇔ если количество простых чисел у текущей суммы кратно 9 и она больше предыдущей, обновляем предыдущую, иначе оставляем s неизменной

На следующем слайде глянем на полное решение задачи...

Рассмотрим полное решение на Python 3:

```
p = lambda x: all(x%j != 0 for j in range(2, int(x**0.5) + 1))
with open('27-82b.txt') as f:
    n = int(f.readline())
    k = [[0, 0]]
    s = 0
    for i in range(n):
        x = int(f.readline())
        combinations = [[a + x, b + p(x)] for a, b in k] + [[x, p(x)]]
        k = {x[1]%9: x for x in sorted(combinations)}.values()
        for a, b in k:
            s = max(a, s) if b%9 == 0 else s
    print(s)
```

#15: №9

Задача, в которой нам придется обрабатывать геометрические прогрессии на каждой итерации. Есть два способа решения, первый - поиск минимальных разностей по остаткам путем допустимых комбинаций, второй - снова дорогой и любимый МЧС. Рассмотрим оба способа:

Набор данных состоит из групп натуральных чисел, каждая группа записана в отдельной строке. В любой группе содержится не менее двух чисел. Рассматриваются все последовательные убывающие геометрические прогрессии с длиной не менее 3, содержащиеся в группах. Из каждой группы следует выбрать геометрическую прогрессию с наибольшей суммой элементов, затем найти максимальную сумму из элементов всех выбранных прогрессий, кратную 6.
Входные данные. Даны два входных файла (файл A и файл B), каждый из которых содержит в первой строке количество чисел N ($2 \le N \le 1000000$).
В каждой из следующих N строк файлов записан сначала размер группы K ($N <= 10$), а затем – K натуральных чисел, не превышающих 10000.

Рассмотрим пример для этой задачи

Пример входного файла:
5
4 216 36 6 1
7 128 64 32 16 8 4 2
3 100 50 25
5 243 81 27 9 3 1
6 72 36 18 9 10
Для указанных входных данных убывающие геометрические прогрессии с длиной не менее 3 имеются в группах: 1 - (216 + 36 + 6 + 1), 2 - (128 + 84 + 32 + 16 + 8 + 4 + 2), 3 - (100 + 50 + 25), 4 - (243 + 81 + 27 + 9 + 3 + 1), 5 - (72 + 36 + 18 + 9). Их сумма равна 1187, не кратна 6. Ответом будет число 1182 - максимальная сумма, кратная 6 получившаяся путём вычитания из общей суммы элементов геометрических прогрессий (из первой группы вычли 1 = 1187 - 1 = 1186, длина прогрессии осталась всё ещё > 2, из четвертой группы вычли два элемента: 3 и 1 = 1186 - 3 - 1 = 1182, длина этой группы тоже осталась > 2.
В ответе укажите два числа: сначала искомое значение для файла A, затем для файла B.

В чем идея решения?

Нам нужно находить геометрические прогрессии. При этом, внимательно читаем условие задачи, мы оттуда выведем следующее: рассматриваются УБЫВАЮЩИЕ прогрессии с ДЛИНОЙ НЕ МЕНЕЕ = 3. Значит, возрастающие прогрессии нас вообще интересовать не будут; прогрессии, состоящие из двух элементов нас тоже интересовать не будут. Кроме того, если в группе прогрессий обнаружено несколько, нас будет интересовать прогрессия с максимальной суммой элементов, об этом тоже сказано в условии. Поэтому, сначала мы должны будем из каждой группы выцепить геометрическую прогрессию (если она имеется) с наибольшей суммой элементов. Для того, чтобы решить эту задачу, напишем соответствующую функцию...

Находим прогрессии с наибольшей суммой

```
def gprog(list): ⇔ функция поиска наибольшей геом. прогрессии (передаем в нее группу)
a = list + [float('inf')] ⇔ добавляем доп. ячейку, чтоб не выходить за границы
b = {0} ⇔ храним тут максимальную найденную прогрессию
for j in range(len(a) - 2): ⇔ проходимся по группе
        if a[j] / a[j + 1] == a[j + 1] / a[j + 2] and a[j] > a[j + 1]: ⇔ если прогрессия убывающая
        g, k = {a[j], a[j + 1]}, 0 ⇔ храним текущую прогрессию
        while a[j] / a[j + 1] == a[j + 2 + k - 1] / a[j + 2 + k]: ⇔ пока коэффициент прогрессии одинаков
            g |= {a[j + 2 + k]} ⇔ добавляем новый элемент, сохраняя текущую прогрессию
            k += 1 ⇔ индекс увеличиваем по циклу
        b = g if sum(g) > sum(b) else b ⇔ если текущая сумма больше, то обновляем b
return b if sum(b) != 0 else 0 ⇔ возвращаем прогрессию с наибольшей суммой элементов
```

Обрабатываем группы чисел

next(f) ⇔ пропускаем первое число (кол-во чисел), оно нам не пригодится

s = 0 ⇔ наша максимальная сумма

while a := ([int(x) for x in f.readline().split()])[1:]: ⇔ читаем группу в list, первое число пропускаем срезом (оно отвечает за кол-во чисел в группе, нам оно не нужно)

if len(a) > 2: ⇔ если в группе хотя бы три элемента (если меньше, нет смысла их рассматривать)

    b = gprog(a) ⇔ находим наибольшую прогрессию в группе

    if b != 0: ⇔ если прогрессия есть

        b = list(b) ⇔ преобразуем в список

        s += sum(b) ⇔ накапливаем сумму на текущую сумму элементов прогрессии

Как найти минимальные разности?

Уже на данном этапе мы имеем максимальную сумму, которую только возможно найти, но, увы, она не делится на 6. Значит, нужно вычесть из нее такой элемент (или такие элементы), которые имеют такой же остаток от деления на 6, как и она, тогда в итоге получим сумму, кратную 6. Эти элементы должны быть минимальны, чтобы сумма осталась максимальной. Кроме того, мы не можем нарушать условие задачи, гласящее нам: “рассматриваются только прогрессии длины не менее = 3”, т. е., мы не можем вычитать элемент, если он хранится в прогрессии длины = 3 (тогда длина этой прогрессии станет 3 - 1 = 2, мы вынуждены будем не учитывать ее полностью), значит, в мин. разности мы можем учитывать не все. Как будем их находить?

Находим минимальные разности

saveost = {0} ⇔ храним тут минимальные разности по остаткам, далее, с каждой итерацией:

if len(b) >= 4: ⇔ если элементов менее 4, то мы не можем ничего вычитать (длина станет менее 3 и мы столкнемся с проблемами)

real = {x + y for x in saveost for y in b} ⇔ находим текущие комбинации сумм из b и предыдущих сохраненных

saveost |= {b[0], b[-1], sum(b[:len(b)-4:]), sum(b[-(len(b)-4)::])} ⇔ добавляем в минимальные разности элементы, входящие в прогрессию, которые можно вычесть (первый или последний, а также суммы элементов, с ограничением до (длина - 4), дабы длина невыбранных элементов составила хотя бы 3 + 1; + 1 берем по той причине, что уже взяли какой-то элемент и скомбинировали его в real

realnow = {x%6: x for x in sorted(real, reverse = 1)}.values() ⇔ сортируем минимальные разности

for k in realnow: ⇔ для разности в realnow

    saveost.add(k) ⇔ сохраняем их в saveost

Получаем результат

На текущем моменте в saveost мы имеем множество разностей с различными остатками. Теперь, чтобы сумма делилась на 6, нужно вычесть из нее минимальную разность с таким же остатком:

```
if s%6 == 0:
print(s)
else:
print(s - min([x for x in saveost if x%6 == s%6]))
```

Рассмотрим полное решение на Python 3:

```python
def gprog(list):
    a = list + [float('inf')]
    b = {0}
    for j in range(len(a) - 2):
        if a[j] / a[j + 1] == a[j + 1] / a[j + 2] and a[j] > a[j + 1]:
            g, k = {a[j], a[j + 1]}, 0
            while a[j] / a[j + 1] == a[j + 2 + k - 1] / a[j + 2 + k]:
                g |= {a[j + 2 + k]}
                k += 1
            b = g if sum(g) > sum(b) else b
    return b if sum(b) != 0 else 0

with open('27-B.txt') as f:
    next(f)
    s = 0
    saveost = {0}
    while a := ([int(x) for x in f.readline().split()])[1:]:
        if len(a) > 2:
            b = gprog(a)
            if b != 0:
                b = list(b)
                s += sum(b)
                if len(b) >= 4:
                    real = {x + y for x in saveost for y in b}
                    saveost |= {b[0], b[-1], sum(b[:len(b)-4:]), sum(b[-(len(b)-4)::])}
                    realnow = {x%6: x for x in sorted(real, reverse = 1)}.values()
                    for k in realnow:
                        saveost.add(k)

print(s) if s%6 == 0 else print(s - min([x for x in saveost if x%6 == s%6]))
```

Рассмотрим решение с помощью МЧС

Автором решения является Елена Драчева.

```
f = open('27-B.txt')
n = int(f.readline())

s = [0]
for _ in range(n):
    a = [int(x) for x in f.readline().split()]
    l = a[0]
    a = a[1:]
    m = []
    for k in range(0, l - 1):
        p = []
        if (a[k] > a[k + 1]):
            p.append(a[k])
            p.append(a[k + 1])
            q = round(a[k] / a[k + 1])
            if (q == (a[k] / a[k + 1])):
                w = a[k + 1] / q
                o=k+2
                while (o<l) and (w==a[o]):
                    p.append(w)
                    o+=1
                    w = w / q
                else:
                    if (sum(p) > sum(m)) and (len(p) > 2):
                        m = p
                    p = []
    if (len(m) > 2):
        su = []
        for i in range(0, len(m) - 1):
            for k in range(i + 1, len(m)):
                if (len(m[i:k + 1]) > 2):
                    su.append(sum(m[i:k + 1]))
        cmb = [a + b for a in su for b in s]
        s = {x % 6: x for x in sorted(cmb)}.values()
```

#15: №10

Гроб, ровно как и прошлая задача, но гроб решаемый. Здесь нам помогут наши друзья-массивы, а еще “обратные” остатки, их прошли в блоках #2 - #4. Гляньте на задачу и подумайте, а как нам учитывать сразу несколько критериев по делимости:

(№ 4411) (Е. Драчева) Дана последовательность из N натуральных чисел. Рассматриваются всевозможные пары различных элементов последовательности, между которыми есть хотя бы одно число, при этом сумма пары кратна трём, а сумма чисел между ними чётна. Найдите количество таких пар.

В чем идея решения?

Нас интересует, чтобы сумма делилась на 3, в то же время, чтобы промежуточная сумма делилась на 2. Значит, мы должны учитывать остатки от деления сразу двух сумм. Для этого мы заведем вложенный массив, который будет иметь размерность: 3 ячейки под 2 дополнительные ячейки. 3 ячейки, соответственно, 3 остатка от деления на 3; а 2 ячейки отвечают за четность и нечетность. Выглядит это так:

k = [[0,0], [0,0], [0,0]] ⇔ храним количество элементов по остаткам

Заведем переменные для хранения суммы и количества:

s = count = 0 ⇔ здесь храним наращиваемую сумму; счетчик

Обрабатываем массивы

xfirst = int(f.readline()) ⇔ читаем первое число вне цикла, далее, проходясь по циклу:

xnext = int(f.readline()) ⇔ прочитали следующее число

s += xfirst ⇔ наращиваем сумму, добавляя предыдущий элемент

count += k[(3 - xnext%3)%3][(2 - s%2)%2] ⇔ увеличиваем счетчик на количество подходящих чисел, подходящие числа определяем по “дополняющим” остаткам, которые позволят образовать в сумме с текущим числом сумму, кратную 3, при этом позволят образовать сумму между этими числами такую, что кратна 2 (тоже берем “дополняющий остаток”. Обратите внимание, что сумму мы нарастили на предыдущий элемент, таким образом мы учли сумму между этими числами, не добавляя к ней текущего элемента. В результате счетчик увеличится на количество подходящих пар, сумма с которыми будет кратна 3, а сумма между которыми будет кратна 2

k[xfirst%3][s%2] += 1 ⇔ обновляем количество чисел и сумм между ними по остаткам

xfirst = xnext ⇔ делаем сдвиг по числу в файлике

Рассмотрим полное решение на Python 3:

```
with open('27-76a.txt') as f:
    n = int(f.readline())
    k = [[0,0], [0,0], [0,0]]
    s = count = 0
    xfirst = int(f.readline())
    for i in range(n - 1):
        xnext = int(f.readline())
        s +=  xfirst
        count  += k[(3 - xnext%3)%3][(2 - s%2)%2]
        k[xfirst%3][s%2] += 1
        xfirst = xnext
    print(count)
```

Как решить неэффективно?

Порой сложно придумать эффективное решение для подобных задач, но один балл все-таки забрать хочется. Забавно, но даже переборное решение этой задачи будет нетипично (нужно немного подумать). Давайте его рассмотрим: в чем будет его идея? Разумеется, для начала запихнем все числа в список, далее будем проходиться по этому списку, но интересным образом: мы будем рассматривать не (j = i + 1), как уже привыкли, а (j = i + 2), ведь между рассматриваемыми парами должно быть хотя бы одно число. Кроме того, мы постоянно будем наращивать сумму, но не от текущего итератора j, а от (j - 1), ведь мы не учитываем текущий элемент. В общем, даже переборное решение интересное, спокойно можно накосячить. Вы сможете с ним ознакомиться на следующем слайде

Рассмотрим неэффективное решение:

```
with open('27-76a.txt') as f:
    n = int(f.readline())
    a = [int(x) for x in f]
    count = 0 #счётчик подходящих пар
    for i in range(n - 1):
        curS = 0 #наращиваемая сумма
        for j in range(i + 2, n):
            curS += a[j - 1] #наращиваем на предыдущий элемент
            if (a[i] + a[j])%3 == 0: #первое условие
                if curS%2 == 0: #второе условие
                    count += 1
    print(count)
```

Рекомендации

Данный материал будет полезен как школьникам, сдающим КЕГЭ, так и преподавателям. Разобраны различные виды 27-х задач, различные способы их решения (как эффективные алгоритмы, так и переборные). Затронута теория по делимости, остаткам, массивам, буферизации, словарям, множествам и даже по переборным алгоритмам.

Очень надеюсь, что материал был полезен, а работа была проделана не зря!

Список использованных источников

Источники задач:

- https://kpolyakov.spb.ru/ - сайт К.Ю. Полякова
- https://rus-ege.sdamgia.ru/ - сайт Решу ЕГЭ
- https://kompege.ru/ - сайт КЕГЭ

Обратная связь

Если у Вас возникли какие-либо вопросы (любого формата: что-то непонятно, с чем-то не согласны, нашли ошибку или, может быть, хотите предложить свое интересное / более простое решение и т. д.), тогда свяжитесь со мной, всячески постараюсь помочь:

- ВКонтакте
- Канал на YouTube
- Адрес Эл. Почты